2020-09

A SEQUENCE-AWARE INTRUSION DETECTION SYSTEM FOR ETHERNET/IP INDUSTRIAL CONTROL NETWORKS

Wetzel, Jonathan L.

Monterey, CA; Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

A SEQUENCE-AWARE INTRUSION DETECTION SYSTEM FOR ETHERNET/IP INDUSTRIAL CONTROL NETWORKS

by

Jonathan L. Wetzel

September 2020

Thesis Advisor:	Thuy D. Nguyen
Co-Advisor:	Marko Orescanin

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2020	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE A SEQUENCE-AWARE INTRUSION DETECTION SYSTEM FOR ETHERNET/IP INDUSTRIAL CONTROL NETWORKS	**5. FUNDING NUMBERS**
6. AUTHOR(S) Jonathan L. Wetzel	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

Industrial control systems (ICS) regulate and monitor critical cyber-physical systems such as the power grid and manufacturing plants. ICS networks are also vulnerable to cyber attacks, and existing defenses against these attacks are similar to those employed by traditional network intrusion detection systems (IDS). However, a typical IDS may not detect semantic attacks on the physical end devices because they follow the protocol specifications to bypass the IDS signatures. Sequence-based attacks, a subset of semantic attacks, can manipulate the ordering of valid commands to cause unsafe conditions for the physical devices. Based on a previous method of detecting sequence-based attacks by using discrete-time Markov chains (DTMC) to model normal ICS network traffic, we implemented a DTMC model for the EtherNet/IP and CIP industrial protocols and observed its effectiveness at recognizing sequence-based attacks. We developed four additional methods for DTMC model creation and compared their ability to detect attacks that the previous method failed to observe. All methods successfully identified attacks causing invalid states or invalid transitions, and only two methods could find localized anomalies. The results confirmed that a DTMC-based sequence-aware IDS could help improve the security posture of national critical infrastructure and Department of the Navy control systems.

14. SUBJECT TERMS industrial control systems, network security, intrusion detection systems, Markov chains	**15. NUMBER OF PAGES** 131
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

A SEQUENCE-AWARE INTRUSION DETECTION SYSTEM FOR ETHERNET/IP INDUSTRIAL CONTROL NETWORKS

Jonathan L. Wetzel
Civilian, CyberCorps – Scholarship For Service
BS, United States Naval Academy, 2011

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2020

Approved by: Thuy D. Nguyen
Advisor

Marko Orescanin
Co-Advisor

Gurminder Singh
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Industrial control systems (ICS) regulate and monitor critical cyber-physical systems such as the power grid and manufacturing plants. ICS networks are also vulnerable to cyber attacks, and existing defenses against these attacks are similar to those employed by traditional network intrusion detection systems (IDS). However, a typical IDS may not detect semantic attacks on the physical end devices because they follow the protocol specifications to bypass the IDS signatures. Sequence-based attacks, a subset of semantic attacks, can manipulate the ordering of valid commands to cause unsafe conditions for the physical devices. Based on a previous method of detecting sequence-based attacks by using discrete-time Markov chains (DTMC) to model normal ICS network traffic, we implemented a DTMC model for the EtherNet/IP and CIP industrial protocols and observed its effectiveness at recognizing sequence-based attacks. We developed four additional methods for DTMC model creation and compared their ability to detect attacks that the previous method failed to observe. All methods successfully identified attacks causing invalid states or invalid transitions, and only two methods could find localized anomalies. The results confirmed that a DTMC-based sequence-aware IDS could help improve the security posture of national critical infrastructure and Department of the Navy control systems.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

σATE	Standard Deviation of Time Elapsed
AFTL	Analog Fluid Tank Lab
ATE	Average Time Elapsed
CIP	Common Industrial Protocol
CPF	Common Packet Format
DIOL	Digital Input Output lab
DST	Destination
DTMC	Discreet-Time Markov Chain
ENIP	EtherNet/IP
FJ	First Jump
FTS	First Time Seen
HMI	Human Machine Interface
ICS	Industrial Control System
ICSICL	Industrial Control System Instructional Cybersecurity Lab
IDS	Intrusion Detection System
I/O	Input/Output
LTS	Last Time Seen
NOP	Normal Operating Procedure
PLC	Programable Logic Controller
SRC	Source

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank my wife, Emilee, for all her support during these last two years and especially the last couple of months. Your motivation and encouragement were greatly appreciated.

I would also like to thank Professors Thuy Nguyen and Marko Orescanin for their guidance and for giving me the opportunity to work on this project.

This material is based upon activities supported by the National Science Foundation under Agreement No 1565443. Any opinions, findings, and conclusions or recommendations expressed are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Industrial control system (ICS) is a term used to describe a category of systems that monitor and control aspects of an industrial process. They are used to regulate power generation and distribution, allow automation of manufacturing processes, and allow operators to quickly respond to unexpected changes in industrial processes [1]. Although ICS is not a new development, advances in networking have led to great levels of standardization within ICS components as well as the desire to connect previously isolated ICS networks with traditional information technology (IT) systems [2]. While this has brought many benefits, it has also made it easier for adversaries to conduct cyber-based attacks that have serious consequences in the real-world. Once such example is the attack in 2015 that resulting in the temporary shutdown on portions of the power grid in Ukraine [3].

This thesis developed four methods to detect potentially destructive attacks against ICS using the EtherNet/IP (ENIP) and Common Industrial Protocol (CIP) communications protocols. Each of these methods is a variation on how a sequence-aware intrusion detection system (IDS) builds observation models to compare against the training model. Sequence-aware IDSs are able to use the information contained network communications to model the operation of the ICS's physical components, and can detect cyber-attacks that would result in damage to the physical components [4]. The network data that contain information about the physical sensors and actuators of an ICS are used as input to a discreet-time Markov chain (DTMC), resulting in a model that reflects the order in which the physical components operate. This observation model can be compared to a model generated from known correct operation, and differences between the two will alert system operators to potential cyber-attacks. We showed how a previous method used to process the observed traffic into the DTMC model can cause the IDS to fail to detect attacks, and proposed four new traffic processing methods that aim to successfully detect as many different types of attacks as possible.

To the best of our knowledge, this work is the first prototype of a sequence-aware IDS for ICS using the ENIP/CIP communication protocols. ENIP/CIP are a pair of protocols developed by ODVA Inc, and are widely used in ICS networks [5]. Our prototype shows that it is possible to develop a DTMC-based sequence-aware IDS for different ICS protocols.

A. MOTIVATION

Like traditional networks, ICS are vulnerable to attacks that exploit flaws in communications protocols and data plane manipulation. However, they are also vulnerable to attacks that place the physical system in an unsafe condition. These sematic attacks [6] require in-depth knowledge of the system being controlled in order to exploit the physical system from the network. A careful sematic attack can manipulate the industrial process to disastrous effect. The most well-known semantic attack is STUXNET, where malware infected the programmable logic controllers (PLCs) in an Iranian uranium enrichment facility and changed the operating speed for the centrifuges, leading to repeated damage of the machinery [7].

Sequence-based attacks, a form of semantic attacks, send valid commands over the ICS network to evade detection, but exploit the ordering of those commands to place the physical system in an unsafe condition [8]. A traditional intrusion detection system (IDS) filters network packets for unusual parameters to identify malicious traffic. However, sequence-based attacks will not use any unusual packets and will therefore not be detected by a traditional IDS [7]. Detecting a sequence-based attack requires a behavior-based intrusion detection system that can understand how each observed network packet will affect the physical system, and compare the observed behavior to the known behavior of the physical system [4]. In [4], Casseli et al. proposed such an IDS, using Discreet-Time Markov Chains (DTMC) to create a model of known commands and the probabilities of how they are ordered. However, it is possible that the implementation of how the observed data is compared to the training model described in [4] could fail to detect attacks that only occur over a short time frame. Additionally, their sequence-aware IDS has only been implemented on networks using the Modbus and IEC 60870-5-104 protocols [4], [9].

B. RESEARCH PLAN

This thesis implements a sequence-aware IDS on an ICS that uses the ENIP/CIP protocols, and proposes four new method of processing observed traffic into the IDS's DTMC model. This was accomplished by implementing the following research plan.

First, an ICS platform for implementing the IDS was identified and the ICS's normal operation observed. Our Industrial Control System Instructional Cybersecurity Lab (ICSICL) was chosen as the target platform. The behavior of the ICSICL's communications between the programmable logic controller (PLC) and human-machine interface (HMI) system was analyzed to develop a parser that extract relevant commands from the observed traffic. Next, a baseline operating procedure for the ICSICL physical components was developed. This was used as the known "normal" operating conditions from which subsequent attacks deviate. Third, a sequence-aware IDS supporting the proposed processing methods was implemented. Fourth, fifteen different test plans were developed, each using different attacks that differed from the normal operating procedure in a different way, and each testing the IDS's ability to detect different types of deviations between the known normal procedure and the attack. Finally, these test scenarios were run, and the IDS's ability to detect each attack was recorded and analyzed.

C. THESIS ORGANIZATION

The remaining chapters are organized as follows. Chapter II provides background information on the ENIP/CIP protocols and the current state of sequence aware IDS implementation. In Chapter III, we discuss the ICS testbed used in this thesis and how its specific implementation of the ENIP/CIP protocols affects our DTMC models. Chapter IV describes the implemented anomaly detection methods, their test plans, and the simulated attacks used to evaluate the effectiveness of each method. Chapter V presents an analysis of the test results. Lastly, Chapter VI summarizes our contributions and discusses possible directions for future research.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND AND RELATED WORK

This chapter provides background on the ENIP and CIP protocols, definitions and history of semantic-based attacks, Markov chain generation, and a summary of previous work relating to using Markov chains to detect attacks against industrial control systems.

A. ETHERNET/IP AND CIP PROTOCOLS

Traditionally, ICS network protocols were created to be optimized for specific applications with severe computing limitations. This resulted in a multitude of network protocols that could execute their intended tasks well but lacked interoperability [10]. However, as the benefits of fully interoperable systems became more apparent, industrial protocols that can run on many different types of network hardware have emerged. The Common Industrial Protocol (CIP), developed by ODVA Inc., is one of these solutions [10]. It is a common application-layer protocol that can be used with four different network stacks as shown in Figure 1. EtherNet/IP (ENIP) provides CIP implementations over traditional TCP/IP networks, and is compatible with both TCP and UDP transport protocols [11]. DeviceNet, CompoNet, and ControlNet are CIP implementations over other proprietary protocols used in industrial control networks.

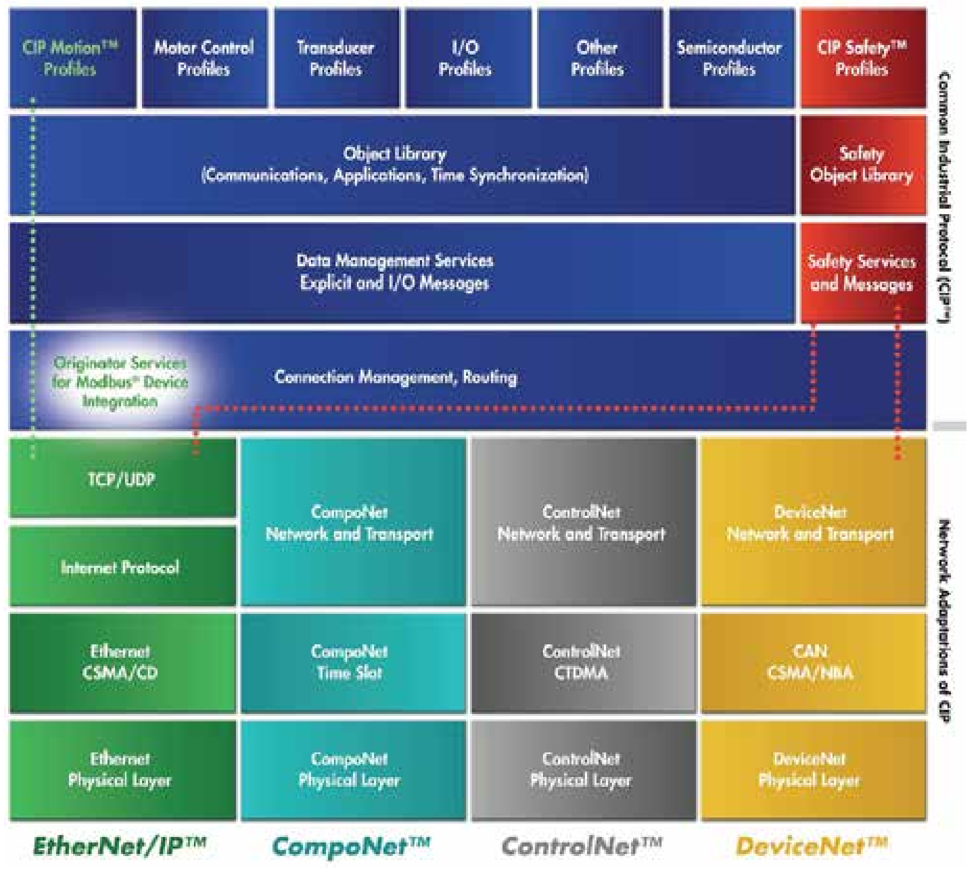

CIP Motion™ Profiles
Motor Control Profiles
Transducer Profiles
I/O Profiles
Other Profiles
Semiconductor Profiles
CIP Safety™ Profiles
Object Library (Communications, Applications, Time Synchronization)
Safety Object Library
Data Management Services Explicit and I/O Messages
Safety Services and Messages
Originator Services for Modbus® Device Integration
Connection Management, Routing
Common Industrial Protocol (CIP™)
TCP/UDP
Internet Protocol
CompoNet Network and Transport
ControlNet Network and Transport
DeviceNet Network and Transport
Ethernet CSMA/CD
CompoNet Time Slot
ControlNet CTDMA
CAN CSMA/NBA
Ethernet Physical Layer
CompoNet Physical Layer
ControlNet Physical Layer
DeviceNet Physical Layer
Network Adaptations of CIP
EtherNet/IP™
CompoNet™
ControlNet™
DeviceNet™

Figure 1. CIP network implementation options. Source: [10]

1. Ethernet/IP Protocol

When implementing CIP on ENIP, the transportation-layer payload consists of an ENIP encapsulation header followed by a variable-length payload that allows for multiple CIP commands to be sent in a single ENIP packet. Figure 2 shows the network stack of an industrial network running ENIP and CIP.

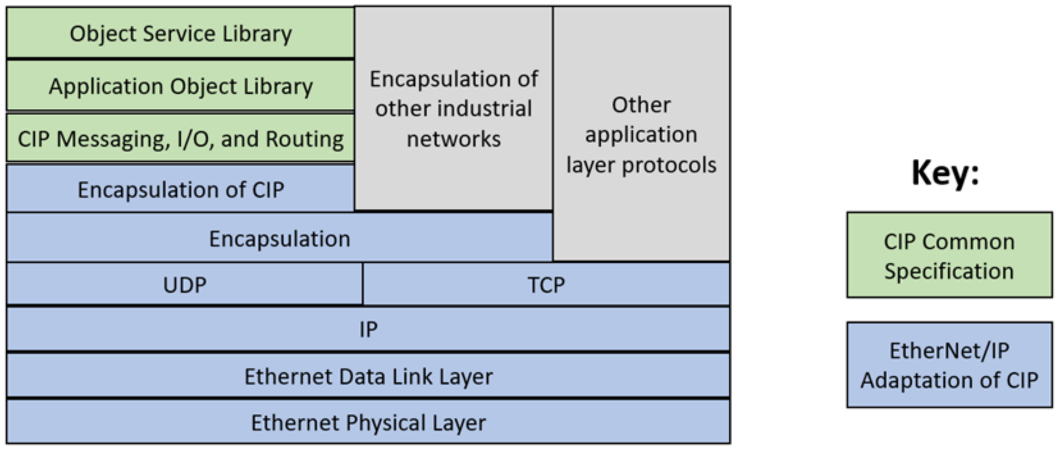

Object Service Library
Application Object Library
CIP Messaging, I/O, and Routing
Encapsulation of CIP
Encapsulation of other industrial networks
Other application layer protocols
Encapsulation
UDP
TCP
IP
Ethernet Data Link Layer
Ethernet Physical Layer
Key:
CIP Common Specification
EtherNet/IP Adaptation of CIP

Figure 2. ENIP and CIP implementation on the network stack

The ENIP protocol provides the important bridge between TCP and CIP by organizing CIP commands in a standard way that can be parsed by any TCP-aware devices. In order to accomplish this, an ENIP packet consists of an encapsulation header that provides basic information about how the encapsulated commands should be handled, and a command-specific payload. The ENIP encapsulation header consists of 24 bytes as shown in Table 1.

Table 1. ENIP encapsulation header. Adapted from [12].

Structure	Field Name	Data Length	Field Value
Encapsulation Headder	Command	2 bytes	Encapsulation command
	Length	2 bytes	Length, in bytes, of the command specific data portion of the message, i.e., the number of bytes following the headder
	Session handle	4 bytes	Session identification
	Status	4 bytes	Status code
	Sender Context	8 bytes	Information pertient only the sender of an encapsulation command. Length of 8
	Options	4 bytes	Options flags
Command Specific Data	encapsulatated Data	Array of 0 to 65511 octet	The encapsulated data portion of the messge. Only requried for certain commands

- Command: Contains one of nine possible ENIP commands, consisting of various ways to set up connection and transmit CIP data. The data captures examined for this thesis exclusively uses the SendUnitData command.

- Length: The number of bytes of the encapsulated data.

- Session Handle: A unique identifier for the ENIP communication session.

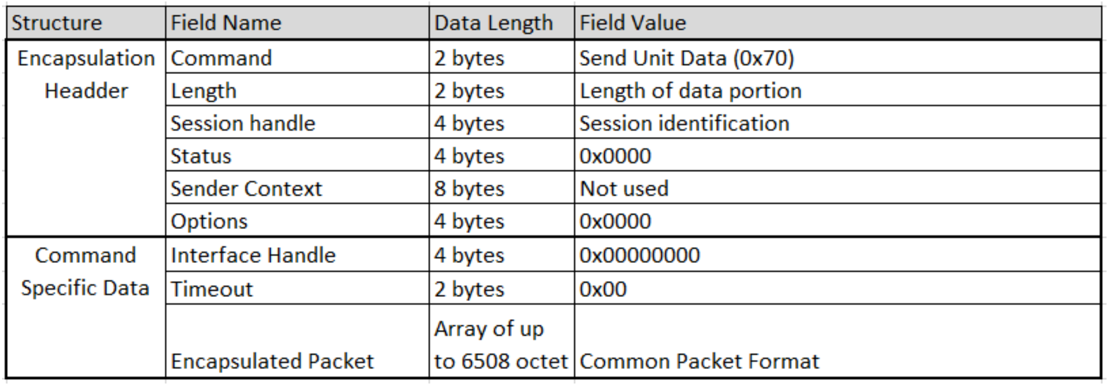
- Status: A value set by the receiver to indicate whether or not the receiver was able to execute the ENIP command. A zero indicates successful execution.

- Sender Context: A unique identifier that can be set by the sender of the command to identify specific elements in a session. Use of this field is optional, and is not implemented in the ICS testbed.

- Options: Command-specific optional flags [12].

The ENIP header does not explicitly distinguish between a request and a reply. This distinction must be made implicitly by observing the ENIP command and its context or explicitly by observing the contents of the encapsulated payload. Table 2 shows the encapsulation header and command-specific values for the SendUnitData command, which is the only ENIP encapsulation command used by the ICSICL system that is relevant for this thesis.

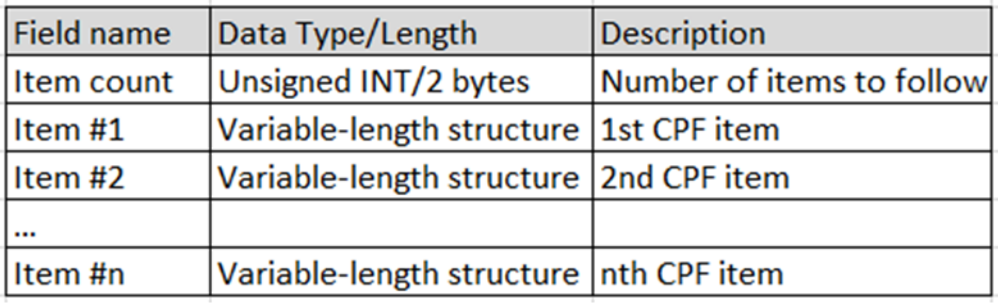
Table 2. Encapsulation header and command data for SendUnitData command. Adapted from [12].



Structure	Field Name	Data Length	Field Value
Encapsulation Headder	Command	2 bytes	Send Unit Data (0x70)
	Length	2 bytes	Length of data portion
	Session handle	4 bytes	Session identification
	Status	4 bytes	0x0000
	Sender Context	8 bytes	Not used
	Options	4 bytes	0x0000
Command Specific Data	Interface Handle	4 bytes	0x00000000
	Timeout	2 bytes	0x00
	Encapsulated Packet	Array of up to 6508 octet	Common Packet Format

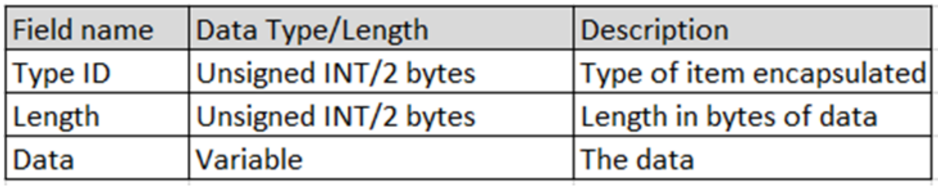
The SendUnitData ENIP encapsulation command type, as well as many others, use the Common Packet Format (CPF) [12], which defines a standard way for encapsulating CIP data. The protocol definition for the CPF header and CPF Item structure are shown in Table 3 and Table 4.

Table 3. Command Packet Format (CPF) header specification. Adapted from [12].



Field name	Data Type/Length	Description
Item count	Unsigned INT/2 bytes	Number of items to follow
Item #1	Variable-length structure	1st CPF item
Item #2	Variable-length structure	2nd CPF item
...		
Item #n	Variable-length structure	nth CPF item

Table 4. CPF item specification. Adapted from [12].



Field name	Data Type/Length	Description
Type ID	Unsigned INT/2 bytes	Type of item encapsulated
Length	Unsigned INT/2 bytes	Length in bytes of data
Data	Variable	The data

The CPF header provides an extendable format for the encapsulation of multiple messages for higher-level protocols. Encapsulated messages are organized into CPF items that provide identification of the encapsulated protocol, data length, and message data. The Type ID field of a CPF item further identifies the purpose of the encapsulated data found in the data section of the CPF item. Type IDs are either address items or data items [12]. An address item provides addressing information for the encapsulated protocol, e.g., whether the protocol is a connection-oriented or not. A data item provides an encapsulation for exchanges of data. The data field of an CFP item contains the encapsulated command for the higher-level protocol, specifically CIP.

2. Common Industrial Protocol

CIP uses object modeling to describe communication services, externally visible behavior, and methods for accessing data. A CIP node is modeled as a set of objects, and objects are structured into classes, instances, and attributes.

A class is a description of a set of the same component type, and an instance is a specific object in that class. Each class has its own set of attributes, and each instance of a class has its own values for the class attributes [13]. The CIP specification [13] includes publicly-defined classes, and any implementations that use these classes must follow the specification. The CIP specification also allows vendors to define their own proprietary classes.

CIP uses the following addressing format to uniquely identify components [13]:

- Node Address: The network address of a CIP node. For ENIP this is the IP address.
- Class Identifier (Class ID): A unique identification value assigned to each object class on the network.
- Instance Identifier (Instance ID): A unique identification value assigned to each instance within a class.

- Attribute Identifier (Attribute ID): A unique identification value for each attribute within an instance or class.
- Service Code: A value that denotes an action request that is directed at a particular object instance or attribute. The service codes for publicly-defined classes can be found in [13].

The CIP specification defines many different class identifiers for objects that are common in an industrial control system, such as types of sensors and actuators as well as connection types with different levels of quality of service [13]. The service codes assigned to a class and the commands and responses associated with those codes define the behaviors of the particular class. However, CIP also enables vendors to use vendor-specific class and service codes to provide custom behavior for their products. The in-house ICSICL testbed makes use of Allen-Bradley/Rockwell Automation ControlLogix Programable Logic Controllers [14] and Rockwell Automation Studio 5000 Logix Designer [14] as the controlling program. The CIP implementation used in the testbed uses vendor-defined classes and service codes for all data transmission, therefore limiting the usefulness of the available CIP documentation for understanding how data is transmitted. Some components of these vendor-defined classes and service codes are discussed in Chapter III.

B. MARKOV CHAINS

A stochastic process is considered a Markov process if in each state, the future state is only dependent on the present state, and independent of any of the past states. This is referred to as the Markov property [15]. If this property exists in a stochastic process that has a finite number of states, it is referred to as a Markov Chain, and if the stochastic process also represents time as a discrete process, it is a Discreet-Time Markov Chain (DTMC). A DTMC consists of a finite amount of states and transitions between those states. Every state must have at least one transition out of that state, but states can transition to themselves. Additionally, each transition has an associated probability the governs the likelihood of that transition being chosen as the process moves to the next state. A common method of representing a Markov chain graphically is via a state-transition diagram. Figure 3 shows a state-transition diagram for a simple DTMC.

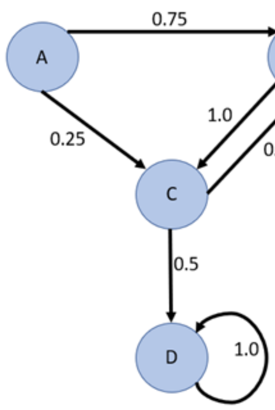
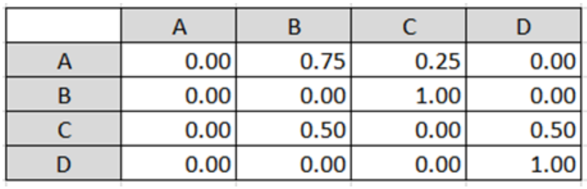


Figure 3. State-transition diagram for a basic DTMC

Each state in a DTMC must have at least one transition leaving that state, and the summation of the probabilities for all transition leaving a given state must be one. In addition to the graphical state-transition diagram, a DTMC can be expressed with a transition probability matrix, as seen in Table 5.

Table 5. Transition probability matric for DTMC shown in Figure 3



	A	B	C	D
A	0.00	0.75	0.25	0.00
B	0.00	0.00	1.00	0.00
C	0.00	0.50	0.00	0.50
D	0.00	0.00	0.00	1.00

A transition probability matrix is a N x N matrix where n is the number of states in the DTMC. Each row in the matrix represents the current state, and the columns represent the possible states to transition to. Every entry shows the transition probability for the transition from I to J, where I is the row and J is the column. Transition probability matrices offer a way to mathematically describe a DTMC, and are used to compare DTMCs among each other [15].

C. SEMANTIC ATTACKS

Like traditional networks, industrial networks are vulnerable to attacks that exploit flaws in communications protocols and data plane manipulation. However, they are also vulnerable to attacks that place the physical system in an unsafe condition. These are called semantic attacks [16].

Semantic attacks require in-depth knowledge of the system being controlled in order to exploit the physical system, but a careful attack can manipulate the industrial process to disastrous effect. The most well-known semantic attack is STUXNET, where malware infected the programmable logic controllers (PLCs) in an Iranian uranium enrichment facility and changed the operating speed for the centrifuges, leading to repeated damage of the machinery [7]. Detection methods for semantic attacks follows the traditional network defense framework of knowing normal activity and then identifying abnormal activity, but instead of focusing on packet construction or communication patterns, they need to understand the underlying physical process and how network traffic can affect that process. One approach is to identify critical setpoints stored in a PLC and monitor attempts to change those values. This can be achieved either by direct knowledge of the plant or by implementing a training phase into the algorithm to identify normal behavior of a category of PLC variables. However, the scalability of this approach is limited due to the significant manual inspection required to identify the data type and memory location of critical variables in proprietary implementation of PLCs [16].

D. SEQUENCE-AWARE INTRUSION DETECTION

A sequence-based attack against an ICS is a type of semantic attack that is designed to cause physical damage to the ICS by causing valid events to occur in an improper order. For example, the command “drain tank #1” could be a valid operation, but it should only be done after a desired flow path has been established. A sequence-based attack could cause the “drain tank #1” command to be sent at an improper time, which could result in tank #1 draining to an undesired location. Making an effective sequence-based attack against an

ICS requires detailed knowledge about the industrial process the ICS is controlling, including knowing which network commands to send at what time to place the underlying process into an unsafe state. However, these commands can be difficult for traditional network monitoring intrusion detection systems to detect because they are well-formed legitimate commands.

Common IDS systems compare packet header and payload data while monitoring against either a black-list of known malicious signatures, or a white-list of acceptable network behavior, i.e., stateful deep packet inspection [6]. This approach is not sufficient for detecting a sequence-based attack, as the attack is delivered by correctly-constructed network packets containing a valid command for one or more of the actuators in the ICS. Therefore, an IDS that can understand the effect of the observed commands on the underlying industrial process, and recognize deviation from the process is needed.

In [4] M. Casseli et al. showed how an IDS could do this by building a DTMC to model the states of the industrial process and transitions between the states. A state S is defined by the 5-tuple <Data, Type, #Elements, FTS, LTS> where:

- Data: uniquely identifies S and describes the portion of information that its events share with each other.
- Type: indicates if S represents elements that include request/response pairs, only request, or only responses.
- #Elements: the number of elements in the sequence that belongs to state S.
- First Time Seen (FTS): the timestamp of the first element in S.
- Last Time Seen (LTS): the timestamp of the last element annexed to S [4].

Every transition δ from a source state (Src) to a destination state (Dst) is defined by a 6-tuple <Probability, Jumps, FJ, LJ, ATE, σATE> where:

- Probability: the ratio of the number of jumps from Src to Dst to the total number of jumps from Src to any only state of the DTMC.

- #Jumps: The number of jumps from Src to Dst in the sequence.
- First Jump (FJ): the timestamp of the first occurrence of this transition.
- Last Jump (LJ): the timestamp of the last occurrence of this transition.
- Average Time Elapsed (ATE): average time between two consequent states of Src and Dst.
- Standard deviation on Time Elapsed (σATE): standard deviation calculated over all occurrence of this transition [4].

A model of correct ICS behavior is then built in the training phase by observing the network traffic of the ICS under normal operating conditions, and either identifying and adding new states and transitions, or updating the existing states and transitions. Once the model is trained it is used for anomaly detection, and is referred to as the baseline DTMC model. In order to detect anomalies, an additional model is trained on the ongoing network traffic which is referred to as the continuously-updating DTMC model. In the detection phase, the baseline DTMC model is dynamically compared to the continuously-learning DTMC model to identify anomalies. There are three possible types of anomalies [4].

1. Unknown State: A state has been created during the detection phase that did not exist in the training phase. This could be the effect of a semantic attack that attempted to change a setpoint to an incorrect value.
2. Unknown Transition: A transition has been created during the detection phase that did not exist in the training phase. This could be the effect of an order-based sequence attack where valid commands are being sent to the actuators, but the sequence of the operations was altered to place the physical system in an unsafe condition. For example, draining a tank before a system is ready to receive the contents of that tank.
3. Unknown Probability: The probabilities of transitions out of a state observed during the detection phase is significantly different than the probabilities seen during the training phase. This could be the effect of a

timing-based sequence-attack, where the repetition (or the lack) of occurrences of two events could cause damage to the physical system [4].

An occurrence of one of these anomalies indicates a possible attack and must be investigated further. There is a reasonable likelihood that false positives are generated and identifying them requires specific knowledge of the communication protocol and the ICS architecture and logic.

III. SEQUENCE-AWARE INTRUSION DETECTION FOR ETHERNET/IP AND CIP

This chapter discusses the in-house industrial control system testbed, the ENIP and CIP communications implemented in the testbed, and the methods used to parse the ENIP and CIP messages to create a DTMC model of the testbed components.

A. TESTBED DESCRIPTION

The Naval Postgraduate School (NPS) has an on-site testbed named Industrial Control System Instructional Cybersecurity Lab (ISCICL) that is designed to allow for security testing on an industrial control system similar to those found in the fleet. The ICSICL testbed was built in 2014 [14], with additional components being added in later years. The core functionality of the testbed consists of an Analog Fluid Tank Lab (AFTL) and a Digital Input/Output Lab (DIOL). The ATFL simulates a tank of fluid as well as the capability to fill and drain the tank. The DIOL consists of three proximity sensors that control the motion of a robotic arm. These components share a Programmable Logic Controller (PLC) and an engineering workstation used to program and supervise the PLC [17]. The engineering workstation also serves as the HMI system.

The PLC runs a ladder logic program that accomplishes three objectives. First, it interfaces with the AFTL to continuously service the tank component sensors and update the local indicator lights. Second, it interfaces with the DIOL to continuously service the proximity sensors and provide serial input that actuates the robotic control arm. Third, it interfaces with the engineering workstation PC to continuously update the development software with the status of the ATFL and DIOL sensors.

1. Analog Fluid Tank Lab (ATFL)

The ATFL simulates an authentic cyber-physical system that controls equipment by monitoring analog Input/Output (I/O) signals. The AFTL local control station is shown in Figure 4.

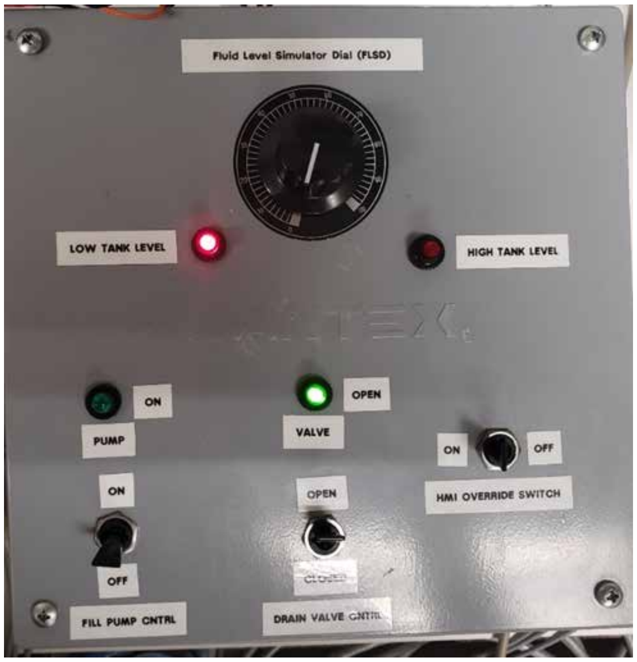


Figure 4. ATFL local control station

The ATFL system was originally designed with two modes, automatic and manual operation. In automatic operation the PLC automatically actuates the fill pump and drain valve to maintain a constant fluid level; however, for this thesis, this mode of operation is not used for any testing. Moving the HMI-override toggle switch to ON changes the ATFL to manual operation mode. In manual operation the PLC continuously monitors the actuators on the ATFL local control station and actuates the status lights on the local station accordingly. The PLC also reports the status of all ATFL components to the engineering workstation. The ATFL contains the following components:

a. Tank-level Rheostat

The tank-level rheostat is a variable resistor controlled by a manual dial. The manual operator can rotate this dial to adjust the simulated level of fluid in the tank. Rotating fully counterclockwise adjusts the tank level to empty and rotating full clockwise adjusts the tank level to full.

b. Tank-level Alarm Lights

The simulated tank level is monitored by the PLC, and if the level is below the low-level setpoint, the PLC will energize the Low Tank Level warning light. Similarly, if the tank level is above the high-level setpoint, the PLC will energize the High Tank Level warning light. The low level setpoint is at ~20% of max capacity, and the high level setpoint is at ~75% max capacity.

c. Fill-pump Toggle Switch and Status Light

The fill pump is controlled by a toggle switch on the front of the AFTL local control station. A pump-status indication light is located directly above the toggle switch. When in manual operation, placing the fill-pump switch to the ON position energizes the status light, but does not automatically change the simulated tank level.

d. Drain-valve Toggle Switch and Status Light

The drain valve is also controlled by a toggle switch on the front of the AFTL local control station. A valve-status indicator light is located directly above the toggle switch. When in manual operation, placing the drain valve to the OPEN position energizes the status light, but does not automatically change the simulated tank level.

e. HMI-override Toggle Switch

The HMI-override functionality is controlled by a toggle switch on the front of the AFTL local control station. Placing the HMI override toggle switch to the ON position enables manual operation of the other actuators on the AFTL local station while removing automatic control of the tank by the HMI system.

The ATFL is connected to the analog input and output modules of the PLC, which, when in HMI- override mode, is programed to regularly send status updates to the HMI system every 0.5 seconds. The HMI system can display the status of the ATFL components as well as the state of the PLC’s internal ladder logic and related variables.

2. Digital Input/Output Lab (DIOL)

The DIOL is intended to simulate an authentic cyber-physical system that controls equipment by monitoring digital I/O signals (Figure 5). The DIOL only operates in manual mode, where the PLC continuously monitors the status of the proximity sensors. When a proximity sensor is tripped, the PLC sends the corresponding command to the robotic control arm. Additionally, the PCL updates the HMI on the status of the proximity sensors every 0.5 seconds. The DIOL contains the following components:

a. Proximity sensors

There are three proximity sensors, labeled sensor 1, sensor 2, and sensor 3. Moving an object close to the top of any of these sensors causes them to trip, causing the digital I/O module to signal the PLC to take the programed action for the tripped sensor. Tripping sensor 1 causes the robotic control arm to move to the raised position. Tripping sensor 2 causes the robotic control arm to move to the lowered position, and tripping sensor 3 causes the arm to move to the secured position as well as removing power to the arm.

b. Robotic control arm

The robotic control arm is a servo-driven commercial arm with three programed positions. It is connected to the PLC via a RS232 serial connection. After receiving a command from the PLC, the arm begins motion and then holds position when it reaches the correct position. However, the arm does not need to complete its motion before another command is received, and the most recent command will override any current action.

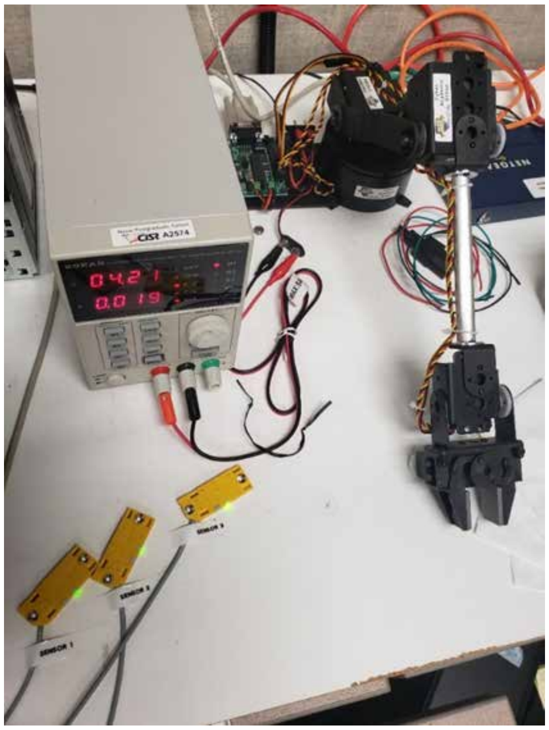


Figure 5. DIOL proximity sensors and control arm (in the secured position)

3. Programable Logic Controller (PLC)

The AFTL and DIOL are controlled by an Allen-Bradley/Rockwell Automation ControlLogix Programable Logic Controller with attached analog I/O, digital I/O and Ethernet (EWEB) modules (Figure 6). The PLC in turn is connected to the HMI via the EWEB module and communicates status updates to the HMI using the ENIP/CIP protocols. The PLC continuously monitors the status of ATFL and DIOL components, and issues output to the local status indicators of the AFTL or the control arm of the DIOL as soon as changes in input are noticed. When communicating with the HMI, the PLC waits until it receives an ENIP Send Unit Data command with an accompanying vendor-specific CIP service request (service code 0x4C) encapsulated within. The PLC responds with a message that contains the current status information of ATFL and DIOL components as the returned data item for the requested service. Since 0x4C is not a standard service code, the syntax and semantic of the response are not described in the CIP protocol specification [13]. The HMI sends this service request approximately every 0.5 seconds. Further details of ENIP/CIP implantation are discussed in Chapter III Section B.

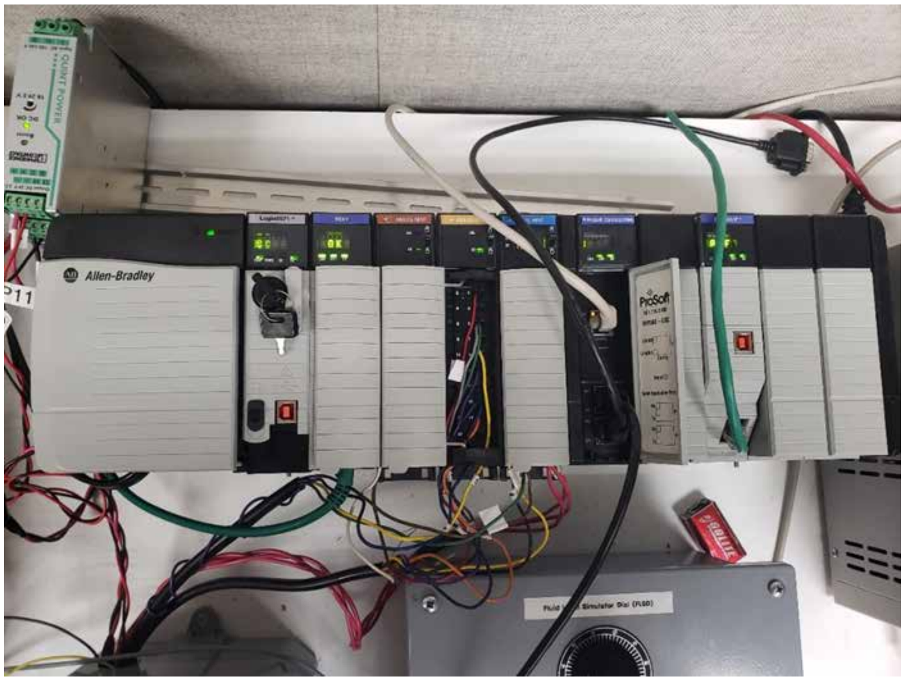


Figure 6. ISCICL Allen-Bradly PLC with attached modules

4. Human Machine Interface

The HMI system consists of a Windows PC running Rockwell Automation Studio 5000 software. It displays the execution status of the PLC ladder logic program and the internal variables representing the monitored components of the AFTL and DIOL. Approximately every 0.5 seconds it sends an ENIP/CIP status request to the PLC, which then responds with the information needed to update the HMI.

B. DATA PREPARATION

The PLC and HMI system are networked with an Ethernet connection and communicate over TCP/IP with ENIP/CIP as the application layer protocol. While Chapter II discussed the basics of the ENIP and CIP protocols, this section discusses how ENIP and CIP are used to monitor and control AFTL and DIOL components. It also describes how the ENIP and CIP messages are parsed to extract position data for these components.

1. HMI-PLC Communications Cycle

The HMI and PLC follow a regular pattern of ENIP/CIP communications, with the HMI cycling through three different forms of the ENIP Send Unit Data command, and the PLC responding to each. Completion of this cycle takes approximately 0.5 seconds. All commands sent by the HMI and responses by the PLC consist of the ENIP encapsulation header associated with the Send Unit Data command, with two CPF items. The first CPF item is always a “Connected Address Item” with a length of 4 bytes. The second CPF item is always a “Connected Data Item,” but the contents and length of this item changes depending on the message being sent. Due to these similarities, messages are referred to by the CIP service request type and length found in the data portion of the ‘Connected Data Item’.

The first message sent in the HMI-PLC communications cycle is a CIP “Get Attribute List” service request with a length of 12 bytes sent from the HMI (packet 1 in Figure 7). The PLC then responds with a 16-byte “Get Attribute List” service response (packet 2 in Figure 7). After the HMI receives the “Get Attribute List” response, it sends a CIP “Multiple Service Packet” Request containing 31 service requests (packet 3 in Figure 7). The services for these service requests are either a vendor-define service code (0x4C) or the service code for the “Get Attribute List” command (0x03). The PLC replies with a “Multiple Service Packet” response containing data for each of the 31 service requests (packet 4 in Figure 7). Once the HMI receives this response, it sends another “Multiple Service Packet” containing nine service requests to the PLC using service codes of 0x4C and 0x03 (packet 5 in Figure 7). The PLC then replies with a “Multiple Service Packet” (packet 6 in Figure 7). Once the HMI receives this response, a communications cycle is complete, and the HMI begins another iteration of the cycle by sending another “Get Attribute List” service request.

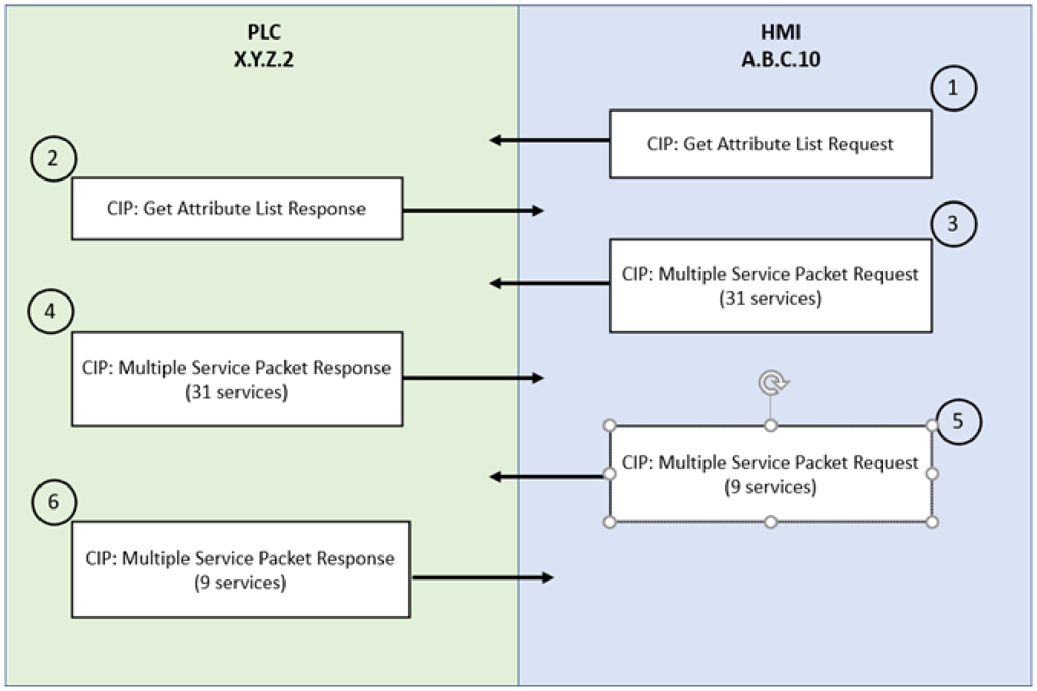

PLC
X.Y.Z.2
HMI
A.B.C.10
1
CIP: Get Attribute List Request
2
CIP: Get Attribute List Response
3
CIP: Multiple Service Packet Request (31 services)
4
CIP: Multiple Service Packet Response (31 services)
5
CIP: Multiple Service Packet Request (9 services)
6
CIP: Multiple Service Packet Response (9 services)

Figure 7. Single iteration of the communications cycle

The next step to being able to extract AFTL and DIOL data for use in a DTMC is to understand which packet within the communications cycle in which the data is contained. The implementation of CIP on the ICSICL system almost exclusively uses vendor-defined classes and service codes; therefore, the CIP specification cannot be used to look up the class and service definitions to understand what information the HMI is requesting during each part of the communications cycle. Therefore, the location of AFTL and DIOL component data was determined by observing packet contents while manipulating the two systems. It was discovered that both the ATFL and DIOL data was contained in the fourth packet in the cycle, which is the "Multiple Service Packet" response containing data for the 31 service requests. Herein, this is referred to as the packet of interest.

2. ENIP Message Structure

To extract the AFTL and DIOL data for use in an IDS, it is necessary to know the location of the data within the packet. Figure 8 shows the relevant components of IP, TCP, and ENIP portions of the packet of interest.

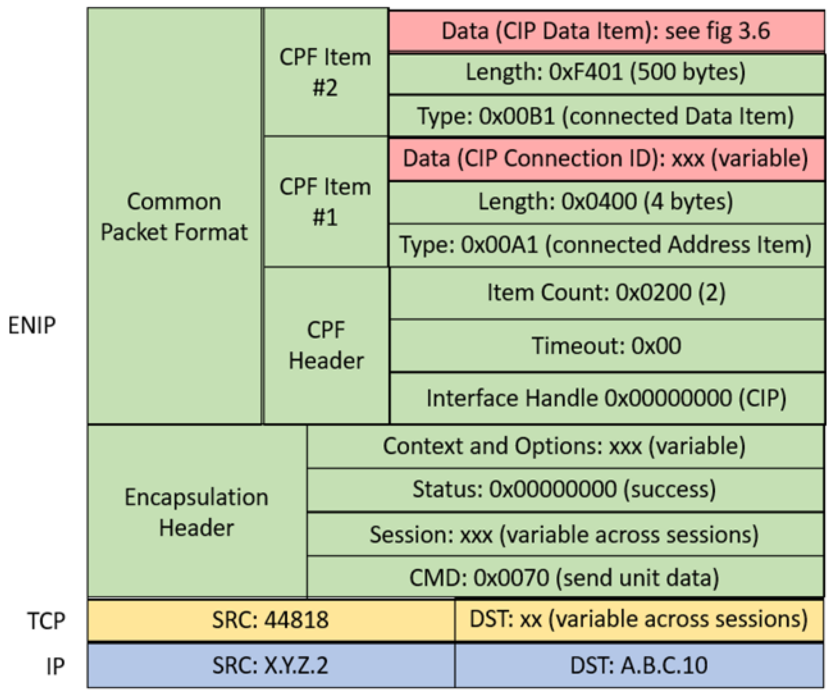
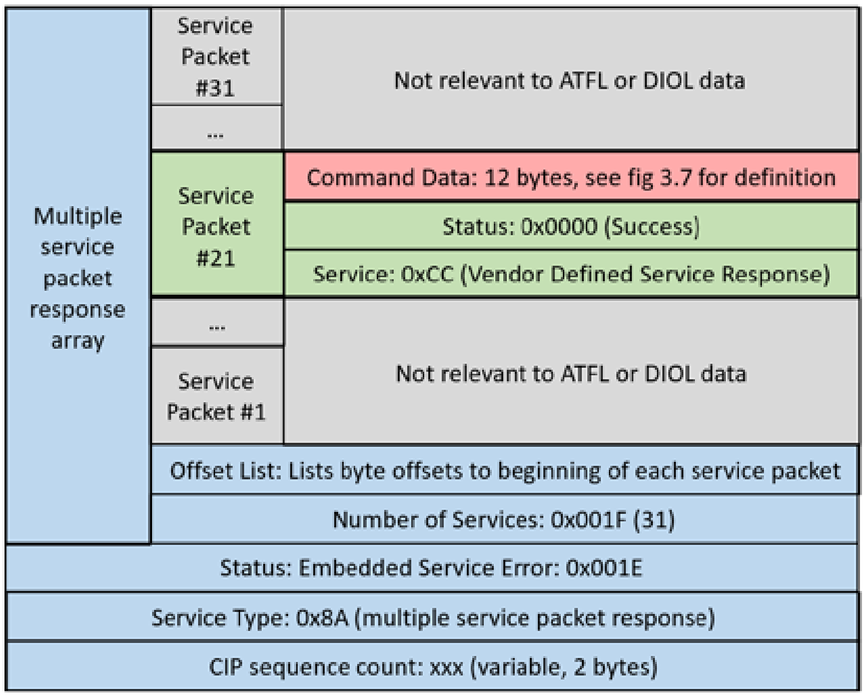

Data (CIP Data Item): see fig 3.6
Length: 0xF401 (500 bytes)
Type: 0x00B1 (connected Data Item)
CPF Item #2
Data (CIP Connection ID): xxx (variable)
Length: 0x0400 (4 bytes)
Type: 0x00A1 (connected Address Item)
CPF Item #1
Common Packet Format
Item Count: 0x0200 (2)
Timeout: 0x00
Interface Handle 0x00000000 (CIP)
CPF Header
ENIP
Context and Options: xxx (variable)
Status: 0x00000000 (success)
Session: xxx (variable across sessions)
CMD: 0x0070 (send unit data)
Encapsulation Header
TCP
SRC: 44818
DST: xx (variable across sessions)
IP
SRC: X.Y.Z.2
DST: A.B.C.10

Figure 8. IP, TCP, and ENIP data in the packet of interest

The IP header in the packet of interest (packet 4 in Figure 7) always has a source address of X.Y.Z.2 (the PLC) and a destination address of A.B.C.10 (the HMI). The TCP port for ENIP is 44818. The TCP payload contains the ENIP message that consists of an ENIP encapsulation header and the associated encapsulation data [12]. For the HMI-PLC communications cycle, the command field in the encapsulation header is always ‘Send Unit Data.’ The encapsulation data portion contains a CPF header and two CPF items. The CPF header consists of the CIP interface handle, a timeout value of 0x00, and an item count of two (0x0200). The first CPF item is a ‘Connected Address Item’ and the second CPF item is a “Connected Data Item.” The ‘Connected Address Item’ contains the CIP connection ID that identifies the message type, and the “Connected Data Item” contains the CIP service request that contains the data being transferred between the PLC and HMI. One feature that distinguishes the packet of interest from the other packets within the communications cycle is the value of 500 bytes in the length field of the “Connected Data Item” element. This observation is used by the parser to identify this packet from the others in the communications cycle, which it then extracts CIP data contained within the “Connected Data Item” element.

3. CIP Message structure

Once the CIP portion of the target packet has been extracted, it must be parsed to locate the AFTL and DIOL data. Observing how this packet's data changes when the AFTL and DIOL are manipulated shows that all the required data is located in the command data portion of the second service packet. Figure 9 shows the format of the CIP data and how to extract the command data portion for service packet #21.



Service Packet #31
Not relevant to ATFL or DIOL data
...
Service Packet #21
Command Data: 12 bytes, see fig 3.7 for definition
Status: 0x0000 (Success)
Service: 0xCC (Vendor Defined Service Response)
...
Not relevant to ATFL or DIOL data
Service Packet #1
Multiple service packet response array
Offset List: Lists byte offsets to beginning of each service packet
Number of Services: 0x001F (31)
Status: Embedded Service Error: 0x001E
Service Type: 0x8A (multiple service packet response)
CIP sequence count: xxx (variable, 2 bytes)

Figure 9. CIP Connected Data Item response for packet of interest

The CIP sequence count is a two-byte value that is similar to the TCP ACK number, and changes with every CIP communication. Service Type is a one-byte value that is always 0x8A for the packet containing the AFTL and DIOL data. 0x8A is the code for a multiple-service-packet response, indicating that multiple CIP service codes are contained within this overall response to a multiple-service request. The next field is the two-byte status field. This field typically indicates that there is an embedded service error code 0x1E00, which is caused by service packets 23–31 returning a partial transfer code. For our analysis, this does not affect the ability of the PLC to communicate the status of the AFTL and DIOL and is ignored.

The next section of CIP data is the multiple-service-packet response array. This is similar to the CPF in the ENIP encapsulation payload as they both allow for an extendable number of data items to share the same protocol header. To allow for ordering and parsing of an arbitrary number of CIP service requests, the first two fields of data within the multiple-service-packet response array are the number of services (2 bytes) and an offset list (two bytes for each service). The offset list contains an entry for each embedded service response; each entry contains a value specifying the number of bytes between the end of the status field and the beginning of a particular service response packet. For example, service response packet #21 is located by 1) finding the start of the offset list, 2) skipping the first forty bytes (the offset value for response packets #1–#20) and reading the next two bytes, which is the offset value for response packet #21, and 3) adding this offset value to the location value at the end of the status field. The resulting value is the starting location of service response packet #21.

We observed that the AFTL and DIOL data is always located in data portion of the service response packet #21. The format of this response consists of three fields. The first field is the service response code, which in this case is always 0xCC indicating a response to the vendor-defined service 0x4C. Next, the two-byte status field is either 0x0000, indicating a successful response, or a non-zero value, indicated an error. During our test we always observed this field was set to 0x0000. The next twelve bytes are the data portion of the response, and contain the status of all AFTL and DIOL components. The data portion was determined to be twelve bytes by calculating the difference between the offset to response packet #21 and the offset to response packet #3.

4. Service Response Data

To build our DTMC model, it is necessary to understand how the 12-bytes data portion of service response packet #21 conveys the status of the AFTL and DIOL components. Figure 10 shows how these bytes are used for the AFTL and DIOL data.

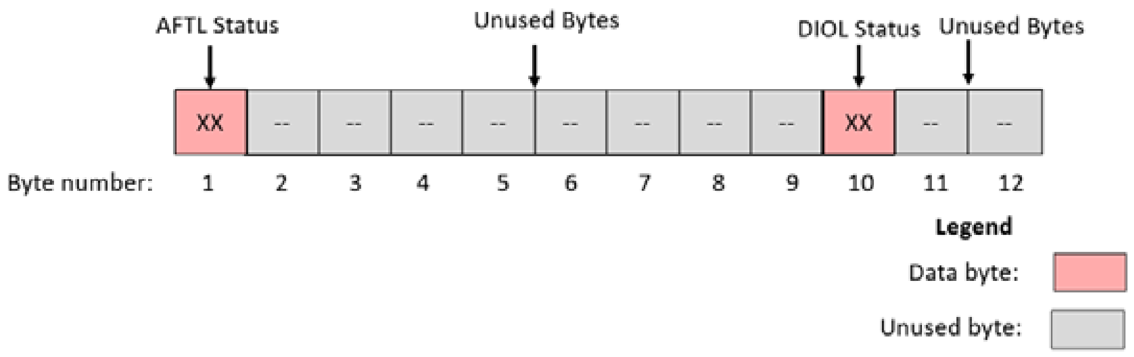


Figure 10. Data portion of service response packet #2

Only two of the twelve bytes are used to report the current state of the AFTL and DIOL components. Byte 1 contains a bitwise representation of the AFTL status, and byte 10 contains a bitwise representation of the DIOL status. All other bytes are unused and have a value of 0x00. Figures 11 and 12 show the bitwise specification of the AFTL and DIOL respectively.

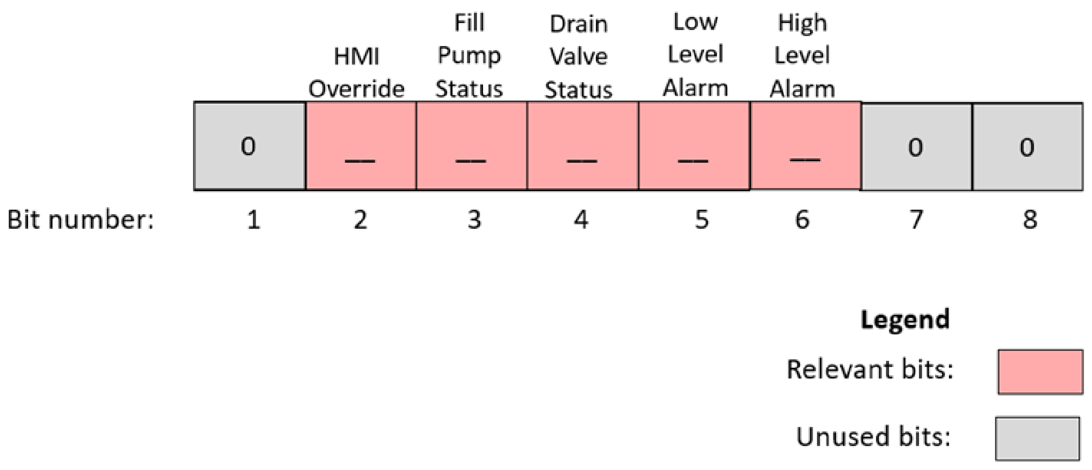


Figure 11. Bitwise specification of AFTL data byte

The AFTL data byte contains a bitwise representation of the five components of the AFTL. Bits 1, 7 and 8 are unused and left a value of zero. Bit 2 represents the status of the HMI-override switch, with a value of 1 representing HMI Override ON, and 0 representing HMI Override OFF. Bit 3 contains the fill-pump status, with pump ON being a value of 1, and pump OFF being a value of 0. Bit 4 contains the drain-valve status, with valve OPEN being a value of 1, and valve SHUT being a value of 0. Bits 5 and 6 are the low-level alarm status and high-level alarm status, with a value of 1 indicating the alarm is ON, and 0 indicated the alarm is OFF. For example, some hexadecimal values frequently seen while operating the AFTL for this byte are:

- 0x58: HMI Override ON, Pump OFF, Valve OPEN, Low Level Alarm IN
- 0x48: HMI Override ON, Pump OFF, Valve SHUT, Low Level Alarm IN
- 0x68: HMI Override ON, Pump ON, Valve SHUT, Low Level Alarm IN
- 0x60: HMI Override ON, Pump ON, Valve SHUT, no level alarms
- 0x64: HMI Override ON, Pump ON, Valve SHUT, High Level Alarm IN
- 0x44: HMI Override ON, Pump OFF, Valve SHUT, High Level Alarm IN
- 0x54: HMI Override ON, Pump OFF, Valve OPEN, High Level Alarm IN
- 0x50: HMI Override ON, Pump OFF, Valve OPEN, no level alarms

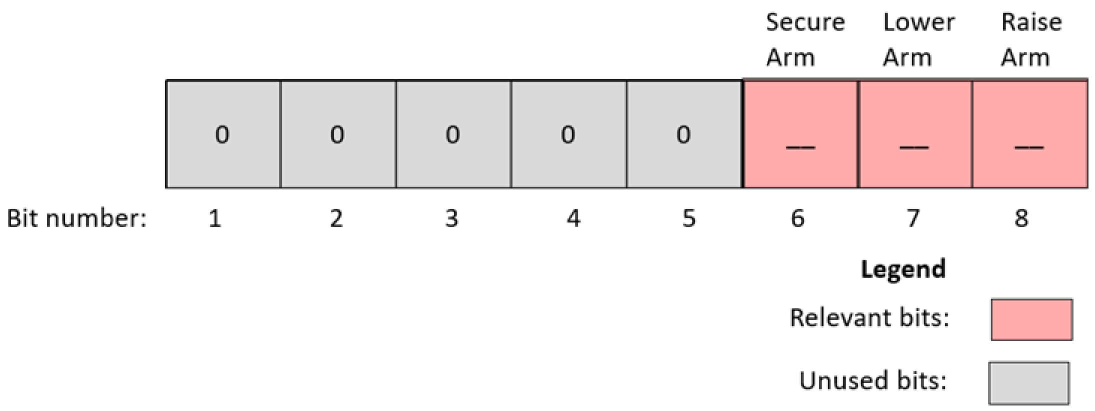


Figure 12. Bitwise specification of DIOL data byte

The DIOL data byte also contains a bitwise representation of the three proximity sensors. Only bits 6, 7, 8 are used, with the rest always being a value of zero. Bit 8 contains the status of sensor #1 (raise arm); when the sensor is tripped the bit 8 is set to 1, otherwise it is 0. Bit 7 represents sensor #2 (lower arm), with a value of 1 if the sensor is tripped and 0 if the sensor is not. Bit 6 represents sensor #3 (secure arm), with a value of 1 if the sensor is tripped and 0 if the sensor is not. For example, some hexadecimal values frequently seen during operation of the DIOL for this byte are:

· 0x02: move arm to lowered position

· 0x01: move arm to raised position

· 0x04: move arm to secured position

C. SEQUENCE-AWARE ATTACK DETECTION OF ENIP AND CIP

The data that represents AFTL and DIOL component status is used as input to a DTMC to model the sequence of operations. As shown in [4], a DTMC can be constructed to model the states of an industrial process and the transitions between them. However, due to how network communications are implemented in ISCICL, the method of defining a states and transitions within the DTMC differs from that used in [4].

1. Model Generator

For this thesis, a state S is defined as a 4-tuple <Data, #Elements, FTS, LTS> where:

· Data: 12 bytes of command data, represented in hexadecimal form.

· #Elements: the number of elements in the sequence that belongs to state S.

· First Time Seen (FTS): the timestamp of the first element in S.

· Last Time Seen (LTS): the timestamp of the last element annexed to S.

The “Type” field used in [4] was removed due to the fact that all relevant ATFL and DIOL data are communicated via a Send Unit Data response from the PLC to the HMI, causing this field to never change and therefore cannot be used to distinguish between different physical state of the AFTL and DIOL.

A transition δ from a source state (Src) to a destination state (Dst) is defined by a 4-tuple <Probability, Jumps, FJ, LJ> where:

- Probability: the ratio of the number of jumps from Src to Dst to the total number of jumps from Src to any other state of the DTMC.
- #Jumps: The number of jumps from Src to Dst in the sequence.
- First Jump (FJ): the timestamp of the first occurrence of this transition.
- Last Jump (LJ): the timestamp of the last occurrence of this transition.

The “Average Time Elapsed” and “Standard Deviation on Time Elapsed” fields used in [4] were also removed. These fields are useful in an ICS that generates a network message when a remote command is issued or a component is changed by modeling the amount of time between relevant events. However, the AFTL and DIOL network traffic consists of a Send Unit Data request from the HMI and subsequent reply from the PLC at regular intervals, resulting in a constant amount of time between observation of new states and subsequently the transitions between those states.

An additional deviation from the methods presented in [4] is the process to select what application layer data is used as input for the DTMC. For protocols and network implementations where each packet represents a new command or status update, the data portion of every observed packet should be used. However, the ICSICL communications cycle is based on the HMI sending the same data request at regular intervals and the PCL responding with data that represents the current system state. Since the data requests from the HMI consist of the same application layer command (CIP Send Unit Data), including them in the DTMC model will prevent the DTMC from modeling state transitions between physical states of the ICSICL components. If the method of using all application level communications as input to the DTMC, as presented in [4], is used on ICSICL, the states associated with the status of AFTL and DIOL components will always transition to a state that represents the HMI asking for a status update. To correct for this problem, the only input to the DTMC model is the PLC data response (packet #4 in Figure 7). With the revised method every DTMC state represents the current configuration of ICSICL components and shows the transitions between component configurations.

2. Transition Matrix

A Python program was developed to parse captured ICSICL traffic and extract the relevant command data. That command data is used as input to our DTMC generator (also a Python program) with the behaviors detailed in Section C.1. As a result, a DTMC that models the status of AFTL and DIOL components is created. Table 6 shows the state transition matrix of our DTMC, Table 7 provides the definitions of the different states in our DTMC, and Figure 13 shows the state transition diagram.

Table 6. DTMC state transition matrix for modeling status changes in ICSICL components

States	0	1	2	3	4	5	6	7	8	9	10
0	0.924	0.075	0	0	0	0	0	0	0	0	0
1	0	0.897	0.102	0	0	0	0	0	0	0	0
2	0	0	0.894	0.105	0	0	0	0	0	0	0
3	0	0	0	0.897	0.102	0	0	0	0	0	0
4	0	0	0	0	0.888	0.111	0	0	0	0	0
5	0	0	0	0	0	0.903	0.038	0.019	0.019	0.019	0
6	0	0	0	0	0	0.15	0.849	0	0	0	0
7	0	0	0	0	0	0.153	0	0.846	0	0	0
8	0	0	0	0	0	0.153	0	0	0.846	0	0
9	0	0	0	0	0	0	0	0	0	0.9	0.1
10	0.1	0	0	0	0	0	0	0	0	0	0.9

Table 7. DTMC State Definitions

State name	Command data hex value	Fill pump status	Drain valve Status	Low-level alarm	High-level Alarm	Robot arm comand
0	0x580000000000000000000000	OFF	OPEN	LIT	NOT LIT	None
1	0x480000000000000000000000	OFF	SHUT	LIT	NOT LIT	None
2	0x680000000000000000000000	ON	SHUT	LIT	NOT LIT	None
3	0x600000000000000000000000	ON	SHUT	NOT LIT	NOT LIT	None
4	0x640000000000000000000000	ON	SHUT	NOT LIT	LIT	None
5	0x440000000000000000000000	OFF	SHUT	NOT LIT	LIT	None
6	0x440000000000000000020000	OFF	SHUT	NOT LIT	LIT	Lower position
7	0x440000000000000000010000	OFF	SHUT	NOT LIT	LIT	Raised position
8	0x440000000000000000040000	OFF	SHUT	NOT LIT	LIT	Secured position
9	0x540000000000000000000000	OFF	OPEN	NOT LIT	LIT	None
10	0x500000000000000000000000	OFF	OPEN	NOT LIT	OUT	None

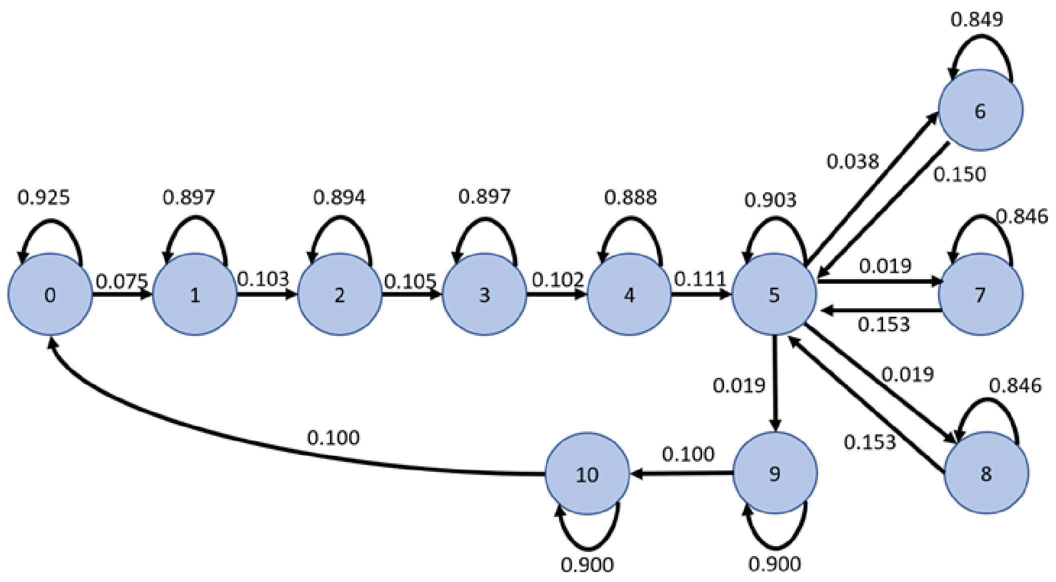


Figure 13. State transition diagram of the DTMC

One interesting property of the state transition matrix is that each state has a high probability of transitioning to itself. This is a result of how status updates are sent in ICSICL, i.e., every 0.5 seconds. If component configuration has not changed between two iterations of the communications cycle, the data sent in the PLC's response will be the same as the previous communication cycle. This results in a majority of the transitions from any one state in the DTMC being a transition back to the same state as shown in Figure 13. Additionally, this also gives a statistical representation of how frequently the components change state. This property is used to identify anomalies in the frequency of ICSICL component manipulations and discussed further in Chapter IV.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. DESIGN AND IMPLEMENTATION

This chapter describes the process used by our prototype DTMC-based IDS to detect sequence-based attacks as well as the series of tests performed to measure performance of the prototype. ICSICL network traffic is captured using Wireshark, and the processes outlined in Chapter III are used to create a DTMC model of the ICSICL component's operation over time. This modeling process is performed twice; first on a traffic capture from known correct operation of the components to create a training model, and second on a traffic capture from the operations that are being monitored for intrusion to create an observation model. The observation model is then compared to the training model for any anomalies.

A. METHODS FOR ANOMALY DETECTION

As shown in [4], there are three types of anomalies a DTMC-based IDS can detect as a response to deviation from the normal sequence of operation in an industrial plant. First, state-based anomalies occur when the states of the observation model do not match the states of the training model. Second, transition-based anomalies occur when the transitions of the observation model do not match the transitions of the training model. Third, probability-based anomalies occur when the probability of the transitions of the observation model are not within the tolerance factor Σ of those in the training model. The exact method of comparing the training and observation model is different for each anomaly. Our prototype implements these three detection methods and four additional proposed methods that are expected to improve anomaly detection performance for attacks that occur over a small period of time.

1. Invalid State Anomaly Detection

An invalid state anomaly exists if the observation model contains one or more states that are not found in the training model. This would occur as the result of an attack that placed the state of the physical ISCICL components in a configuration that is not present in the training model. This type of anomaly is detected by iterating through each state in the observation model and checking if that state is also present in the training model. If it is not, our detection algorithm considers it as an invalid state anomaly. Figure 14 shows a comparison between a training and observation model that results in an invalid state anomaly.

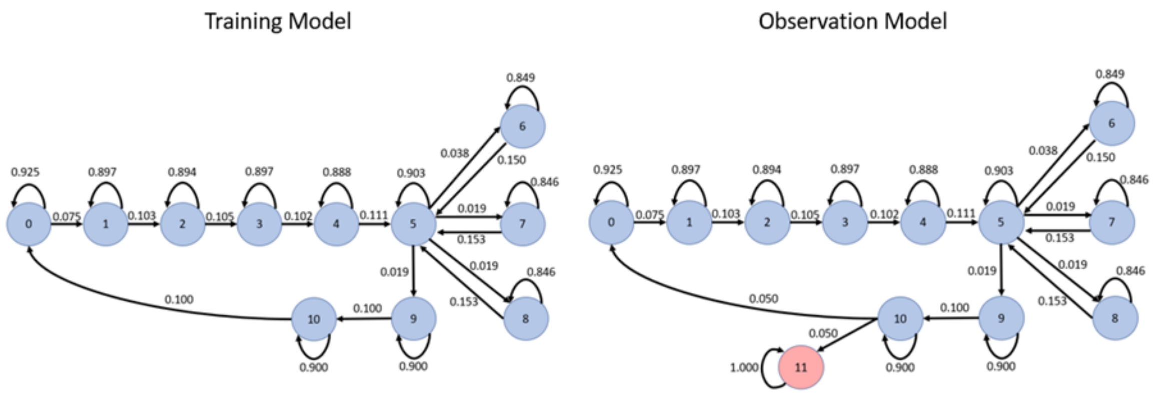


Figure 14. Model comparison resulting in an invalid state anomaly

2. Invalid Transition Anomaly Detection

An invalid transition anomaly exists if the observation model contains one or more transitions of a particular state that are not found in the training model. This would occur as the result of an attack that manipulated the physical ICSICL components between two valid configurations, but the transition from the first to the second configuration was not present in the training model. This type of anomaly is detected by iterating through each transition of each state in the observation model and checking if that transition is also present in the training model. If it is not, our algorithm treats it as an invalid transition anomaly. Figure 15 shows a comparison between a training and observation model that results in an invalid transition anomaly.

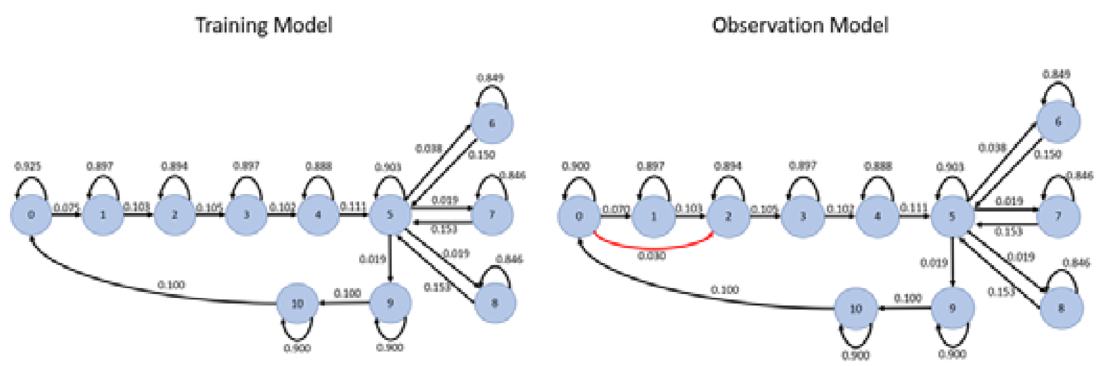


Figure 15. Model comparison resulting in an invalid transition anomaly

3. Invalid Probability Anomaly Detection: Batch Processing

An invalid probability anomaly exits if there are one or more transition probabilities in the observation model that differ than the corresponding transition probabilities in the training model by more than the probability tolerance factor Σ. This is the most difficult type of attack to detect because both the start and end states as well as the transition between them are found in the training model. This means the attack will not place the system in a new condition (resulting in a new state in the model) or incorrectly transition the system between known conditions (resulting in a new transition in the model). Instead, this attack results in a significant change in the probabilities of what transition is chosen when exiting a given state. An example of an attack that could do this is one that repeatably cycles a component instead of allowing the underlying process to move on. If a limited amount of component cycling is allowed, this attack would not result in the creation of new states or transitions in the generated model. Instead, it would alter the probabilities of the transitions leaving the states that represent each position of the component that is being cycled. The probability of these states transitioning to each other would increase, and the probability of a transition to the rest of the known states would decrease. Additionally, because ISCICL status updates occur at a constant interval, an attack that significantly altered the amount of time between component manipulations would also result in an invalid probability anomaly. Figure 16 shows a comparison between a training and observation model that results in an invalid probability anomaly.

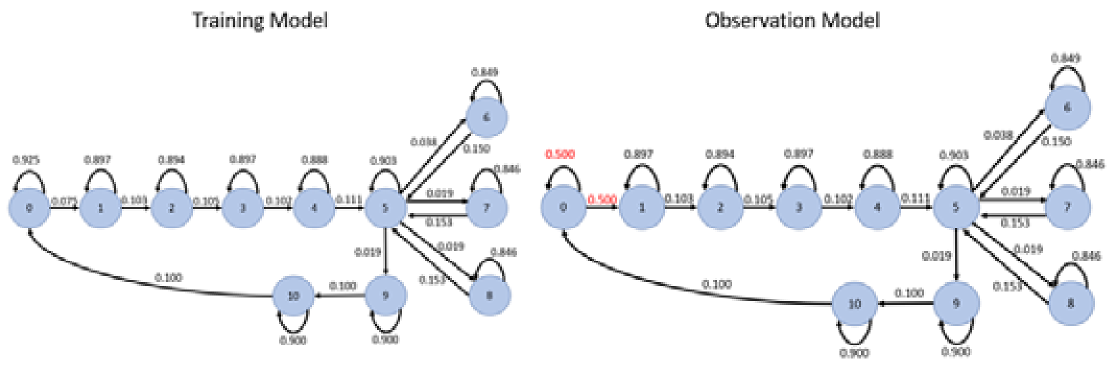


Figure 16. Model comparison resulting in an invalid probability anomaly

The naïve method of detecting invalid probability anomalies is called batch processing. Batch processing involves creating a single training model using every packet in the training capture, and then comparing the difference between each transition probability in the training and observation models to ensure it is less than Σ. If the absolute value of the difference is found to be greater than Σ, an invalid transition probability anomaly is found. The method used for setting the initial value of Σ is discussed in Chapter IV Section D. Figure 17 shows a block diagram of the batch processing method.

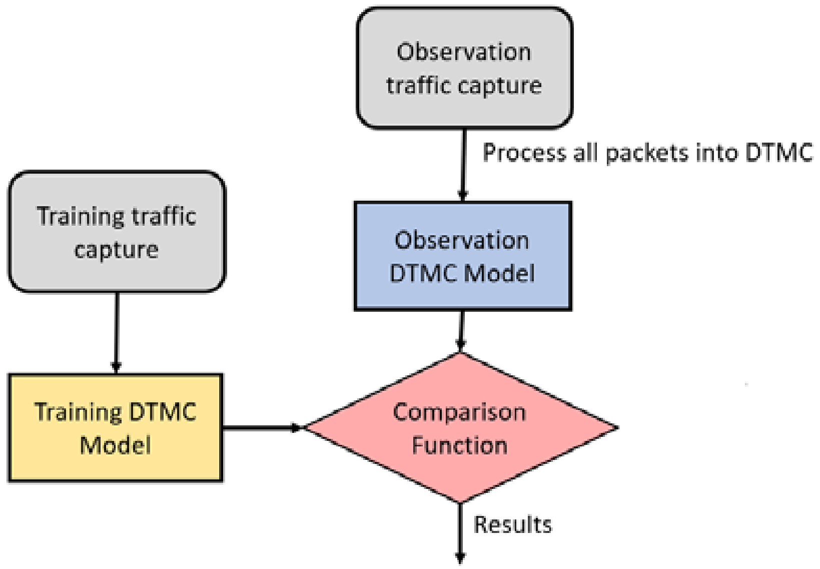


Figure 17. Batch processing method for transition probability anomaly detection

For all five probability anomaly detection methods supported by our prototype, the training model is created by reading the entire training traffic capture into a DTMC model, as shown in Figure 17.

4. Localized transition probability deviations

The batch method of transition probability detection is suitable if the attack occurs throughout the course of the captured traffic. However, if the attack only occurs over a relatively small timeframe its effects on the final transition probabilities can be very small and cause the attack to go undetected. Figure 47 in Chapter V Section 3 shows how batch processing can fail to detect attacks in this manner. To address this issue, our prototype also implemented four additional methods of transition probability anomaly detection.

a. Stream Processing

Stream processing limits the number of packets added to the observation model at a single time. A counter is maintained as each packet is processed into the DTMC observation model, and once the interval size X has been reached the observation model is compared with the training model for any anomalies. After this comparison, the counter is reset and the processing of data from the observation captures resumes. This process repeats until all packets have been processed into the observation model. With this method, the observation model is check for anomalies every X (interval size) packets, as opposed to only checking for anomalies once every packet has been processed. If an anomaly is detected at any of the comparisons the overall process is considered to have detected an anomaly.

It is predicted that this method allows for higher likelihood of detecting localized anomalies at the beginning of the packet capture. However, as the total size of the observation DTMC grows, it is expected that performance will converge to be the same as the batch processing method. Additionally, the interval size must be carefully chosen, as an interval size that is too large will reduce the capability of localized anomaly detection, and an interval size that is too small can result in false positives due to the full set of possible transitions not being processed into the DTMC. Chapter IV Section D discusses how an initial value for interval size is determined. Figure 18 shows a block diagram of the stream processing method.

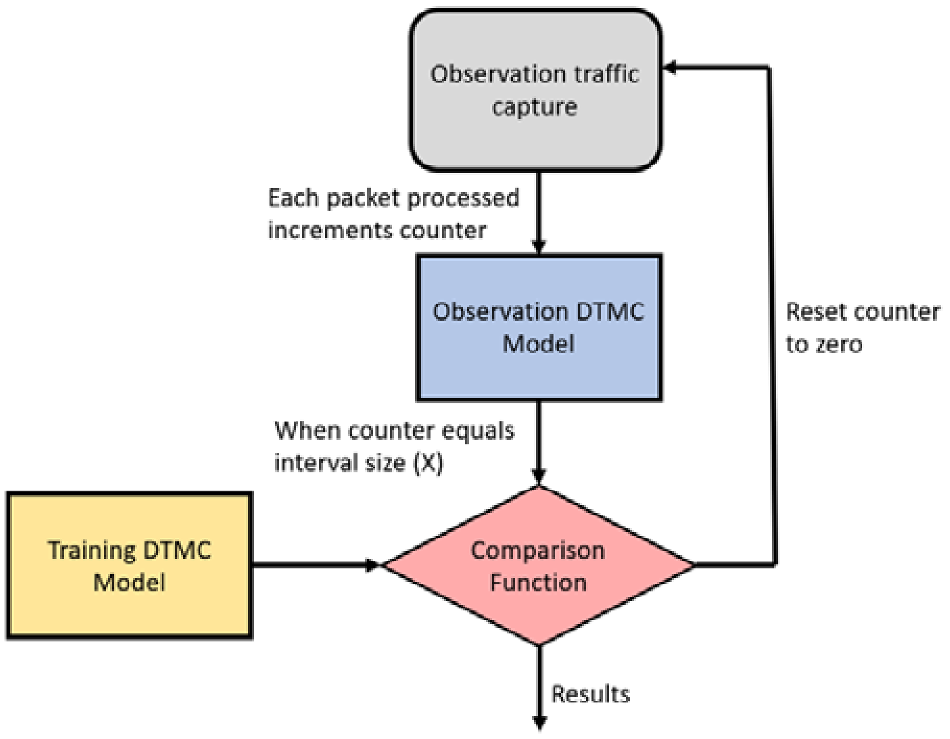

Observation traffic capture
Each packet processed increments counter
Observation DTMC Model
Reset counter to zero
When counter equals interval size (X)
Training DTMC Model
Comparison Function
Results

Figure 18. Stream processing method for transition probability anomaly detection

b. Repeat Processing

Repeat processing creates interim observation models as each packet from the observation traffic capture is processed. Similar to stream processing, this method uses a counter to limit the number of packets added to a particular observation model, and once the counter has reached the interval size, the observation model is compared to the training model for anomalies. Unlike stream processing, after the comparison and the interval size counter is reset, the current observation model is discarded and a new generation of observation model is created. This process repeats until there are no unprocessed packets in the observation traffic capture, and the final generation of observation model is compared with the training model.

This method is expected to be able to detect localized anomalies that occur throughout the observation capture and should not suffer the fall-off in performance that is expected from the stream processing method. However, because the number of packets used to the create the final observation model is not constant, it is possible that the full range of possible state, transitions, and transition probabilities are not represented in that model. This will possibly cause false positives in the final comparison and decrease the overall performance of the model. Additionally, it is expected that transition probability anomalies that occur in related packets that are separated by the interval size are less likely to be detected. The likely cause of this is the distribution of the packets containing the evidence of the anomaly being distributed across two observation models, decreasing the magnitude of the change of the transition probability of concern. Figure 19 shows a block diagram of the repeat processing method.

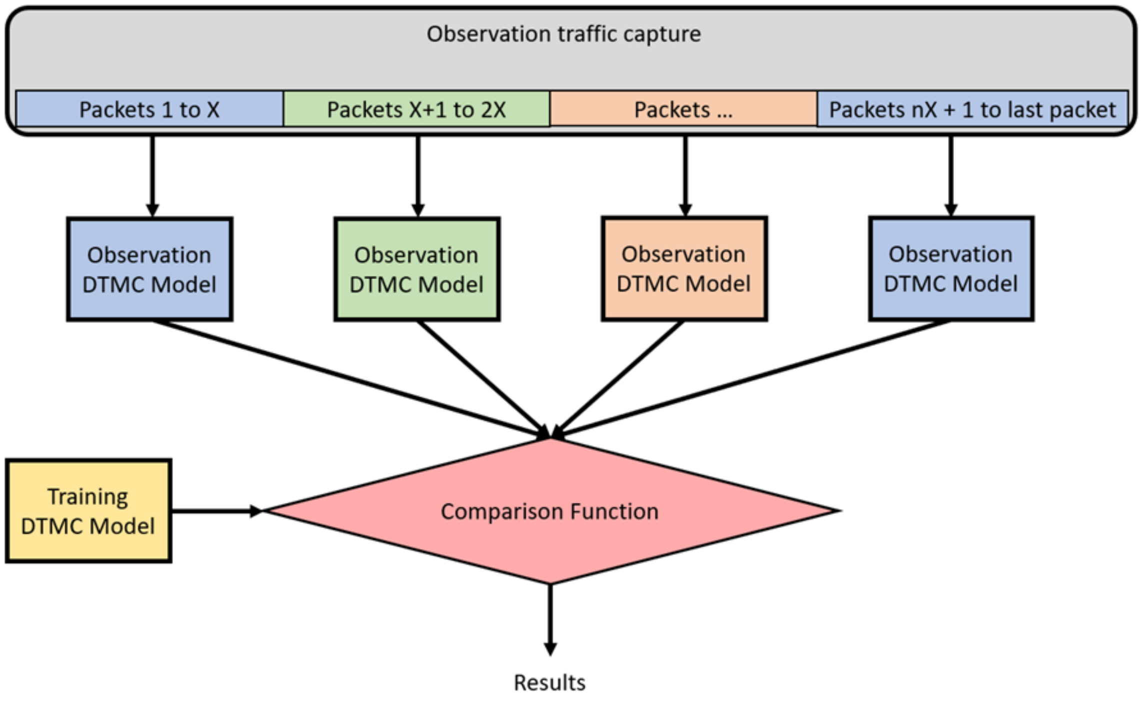


Figure 19. Repeat processing method for transition probability anomaly detection

c. Sliding Window Processing

Sliding window processing is a modification to repeat processing that includes overlap between the packets used to create consecutive generations of observation models. Once the first X (interval size) packets have been processed into a DTMC observation model and compared with the training model, the old observation model is discarded, and a new generation of model is created. However, instead of starting model development at the next unprocessed packet, the overlap size (Y) is used to select the last Y packets processed in the previous model and include them in development of the current generation's model. For example, if interval size is 100 and overlap size is 50, the first observation model will be generated from packets 1 – 100, and the second observation model will be generated from packets 51–150. The third observation model will then be generated from packets 101–200. This process repeats until there are no unprocessed packets in the observation traffic capture, and the final observation model is compared with the training model. The overall result is that many separate observation models are generated from the observation capture, with each subsequent model re-using Y (overlap size) number of packets from the previous model. The expected performance of this method is similar to the expected performance of the repeat processing method, with the potential performance improvement of being more likely to detect probability anomalies that occur over the transition from one interval size to another. Using a sliding window to overlap what observation packets are used to generate subsequent observation models decreases the likelihood that the effect of an anomaly is separated between two observation models. However, the issue of an unreliable final model is still present. Chapter IV Section D discusses how an initial value for overlap size is determined. Figure 20 shows a block diagram of the sliding window processing method.

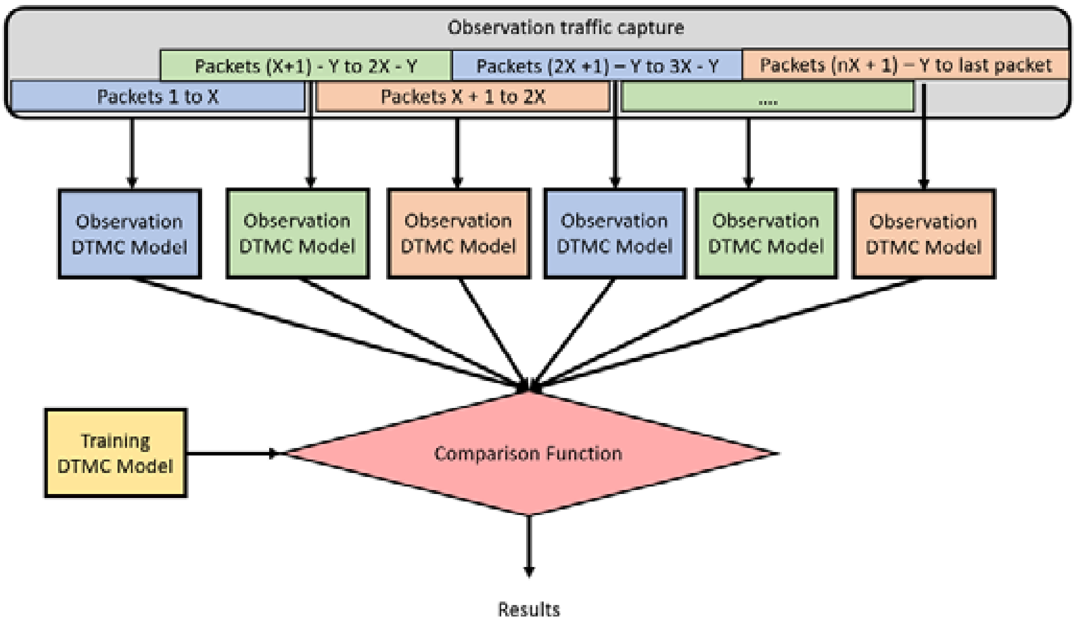


Figure 20. Sliding window processing method for transition probability anomaly detection

d. Alpha Filter Processing

The Alpha filter method is similar to repeat processing except that once an observation DTMC is used in a comparison function it is not totally discarded. Instead, it is used as input to an alpha filter to modify the next generation's observation model's transition table before comparing to the training model. This method is also known as exponential smoothing. The alpha filter processing method consists of the following calculation:

$$TMatrix_{flitered}[t] = \alpha \times TMatrix_{filtered}[t-1] + (1-\alpha) \times TMatrix_{raw}[t] \quad (1)$$

Where:

- $TMatrix_{filtered}$ is a transition matrix of the DTMC model that has been passed through the alpha filter.
- $TMatrix_{raw}$ is a transition matrix of the DTMC model that has not been passed through the alpha filter.

- α is the filter constant. Must be withing range [0:1].
- t is the number of $TMatrix_{raw}$ that have been generated.

The result is that the transition probability values found in a given filtered model are a weighted average of the current model's unfiltered probability values and the previous model's probability values. The filter constant α determines the weight given to the previous model's probability values. If α is set at 0.5, both the current raw model and previous filtered model have an equal effect on the current filtered model's transition probability values. As α approaches zero, the previous filtered model has less effect on the current filtered model, and as α approaches one, the previous filtered model has more effect on the current filtered output. This method is expected to retain the localized anomaly detection benefits of the repeat processing and sliding window processing methods while reducing the number of false positives generated by the final observation model seen in those processing methods. Figure 21 shows a block diagram of the alpha filter processing method.

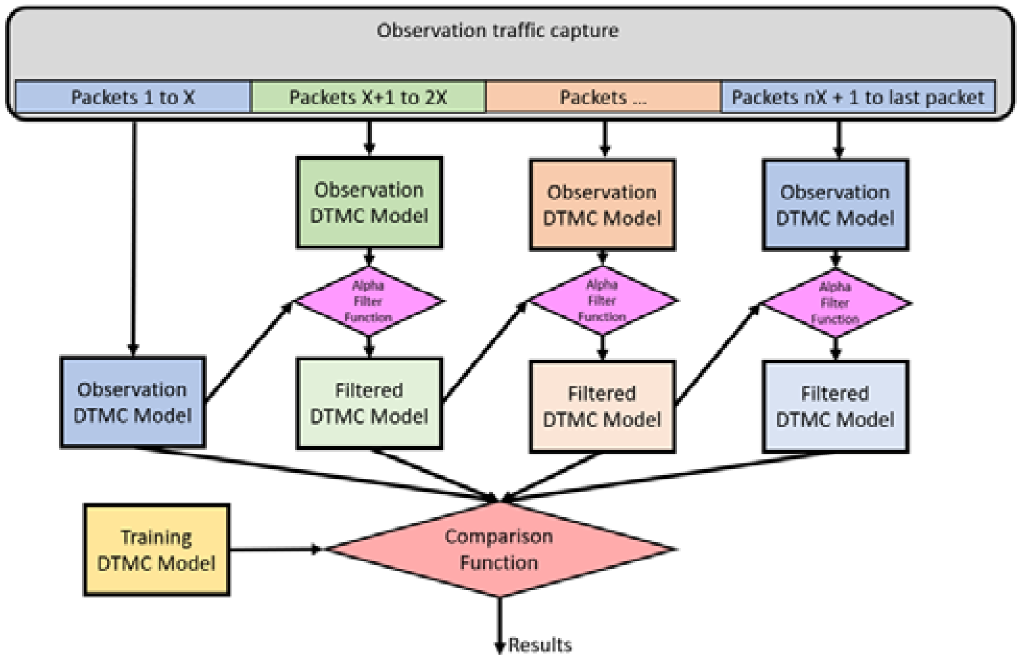


Figure 21. Alpha filter processing method for transition probability anomaly detection

B. BASELINE OPERATING PROCEDURE

In order to observe the performance of anomaly detection techniques, a baseline of normal operations on the industrial system being observed must be defined. This section defines the normal operating procedure of the ICSICL components used to create the training DTMC model. This operating procedure simulates a continuous process found in many industrial applications. The requirements used to design the Normal Operating Procedure (NOP) are as follows:

1. It must be cyclic, with the initial condition of the components the same as the final condition. This will allow the procedure to be repeated an arbitrary number of times.
2. It must be possible to generate an invalid state anomaly. Therefore, the full state space of possible component positions cannot be used, as there must be a component configuration not seen by the training model.
3. It must be possible to generate an invalid transition anomaly. Therefore, there needs to be a pair of component positions that are one manipulation different from each other, but the normal operating procedure cannot contain a direct transition between those positions.
4. It must be possible to generate an invalid transition probability anomaly. Therefore, the normal operating procedure must allow for some amount of cycling between two component positions, but excessive cycling should be defined as a violation of the procedure.

For our experiments, an operator manually manipulates the components at the AFTL and DIOL according to the procedure to simulate an automated industrial process. While writing a separate ladder logic program for the PLC and HMI system to automatically perform the same process is possible, doing so would require a much more precise understanding of the vendor-defined CIP services and classes used in the ICSICL network, which is outside the scope of this thesis.

1. Initial Condition

Prior to starting the normal operating procedure and the training traffic capture, the AFTL and DIOL components of the system must be placed in the correct initial conditions as shown in Table 8. The normal operating procedure described in Section B.2 returns all AFTL and DIOL components to the initial conditions, allowing another iteration of the normal operating procedure to be performed immediately after finishing the previous iteration.

Table 8. Initial conditions for ATFL and DIOL components

Component	Initial Condition
Tank low-level light	LIT
Tank-level control knob	Rotate fully counterclockwise
Drain valve	OPEN
Pump control	OFF
Control arm	SECURED
HMI override	ON

Although having the HMI-override switch set to ON is not typical for most industrial applications, it is required by ICSICL to allow the AFTL local control station to manually manipulate the ATFL components.

2. Normal Operating Procedure

After the AFTL and DIOL components are placed in the initial conditions and the training traffic capture is started, the component manipulations shown in the normal operating procedure below can begin.

1. Drain valve > SHUT
2. Pump control > ON
3. Tank level control knob -> rotate clockwise to 12-o’clock position (Tank low level light > OFF)

4. Tank level control knob -> rotate fully clockwise (Tank high level light -> ON)

5. Pump control > OFF

(a) Pump control > ON

(b) Pump control > OFF

Note: Steps 5a and 5b may be performed between zero and two times. If 5a is performed, 5b must be performed before continuing to step 6

6. Control arm > LOWERED position (hold sensor 2 for five seconds)

7. Control arm > RAISED (hold sensor 1 for 5 sec)

8. Control arm > LOWERED (hold sensor 2 for 5 sec)

(a) Control arm > RAISED (hold sensor 1 for 5 sec)

(b) Control arm > LOWERED (hold sensor 2 for 5 sec)

Note: 8a and 8b may be performed between zero and two times. If 8a is performed, 8b must be performed before continuing to step 9

9. Control arm > SECURED position (hold sensor 3 for 5 sec)

10. Drain valve > OPEN

11. Tank level control knob > rotate counterclockwise to 12-o'clock position (Tank high level light > OFF)

12. Tank level control knob > rotate fully counterclockwise (Tank low level light > ON)

Additionally, the operator must wait five seconds in between performing each step. This is to standardize the number of packets generated for each configuration of ICSICL components. This requirement extends to the operation of the proximity sensors in steps 6 – 9 because the DIOL status updates described in Chapter 3 only occur if the sensor is pressed. Therefore, pressing for five seconds when activating the sensors will ensure updates from the PLC to the HMI include consistent sensor statuses.

At steps 5 and 8, a limited amount of component cycling is allowed. Steps 5a/5b and 8a/8b can only be performed up to two times; any further cycling of those components is defined as out of specification. The two-cycles limit was chosen to meet requirement #4 while keeping the number of allowed cycles low to facilitate ease of testing.

3. Generation of Training Models

Using the NOP, four sets of traffic captures are generated to create four training models. These four captures differ in the amount of component cycling (steps 5a/5b and 8a/8b) they include, but all remain within the specification defined in the NOP. These traffic captures are referred to as NOP1–NOP4 captures, and their models referred to as NOP1 – NOP4 training models, respectively. Table 9 defines NOP1-NOP4.

Table 9. Definition of normal operating procedure traffic captures

Capture Name	Number of iterations of NOP	Special instructions
NOP1	5	Do not perform steps 5a/5b or 8a/8b
NOP2	5	Repeat steps 5a/5b twice
NOP3	5	Repeat steps 8a/8b twice
NOP4	5	Repeat steps 5a/5b and 8a/8b twice

While each of the NOP captures is used to calculate an initial value for the probability tolerance factor Σ (described in Section 4.3), only NOP4 is used to create training models used for attack detection because it is the only one that includes the cycling of both components described in steps 5a/5b and 8a/8b. Hence, the NOP4 model is the only model to contain transitions and probability adjustments that represent these cycles. Using a NOP model that does not include the allowable amount of cycling of these two components as a training model would cause 1) false positive transition anomalies if compared to a model made from a capture that contains the allowable amount of cycling of the fill pump (steps 5a/5b), or 2) false positive probability anomalies if compared to a model made from a capture that contains the allowable amount of cycling of the robotic arm (steps 8a/8b) . For example, when NOP1 is used as the training model, comparing it to an observation model generated by NOP2 or NOP4 will result in an invalid transition anomaly. Figures 22 and 23 show the state diagram and transition matrix for each of the NOP models. The orange highlighted transition shows the effect of performing steps 5a/5b in NOP2 and NOP4, and the green highlighted transition probabilities shows the effect of performing steps 8a/8b in NOP3 and NOP4.

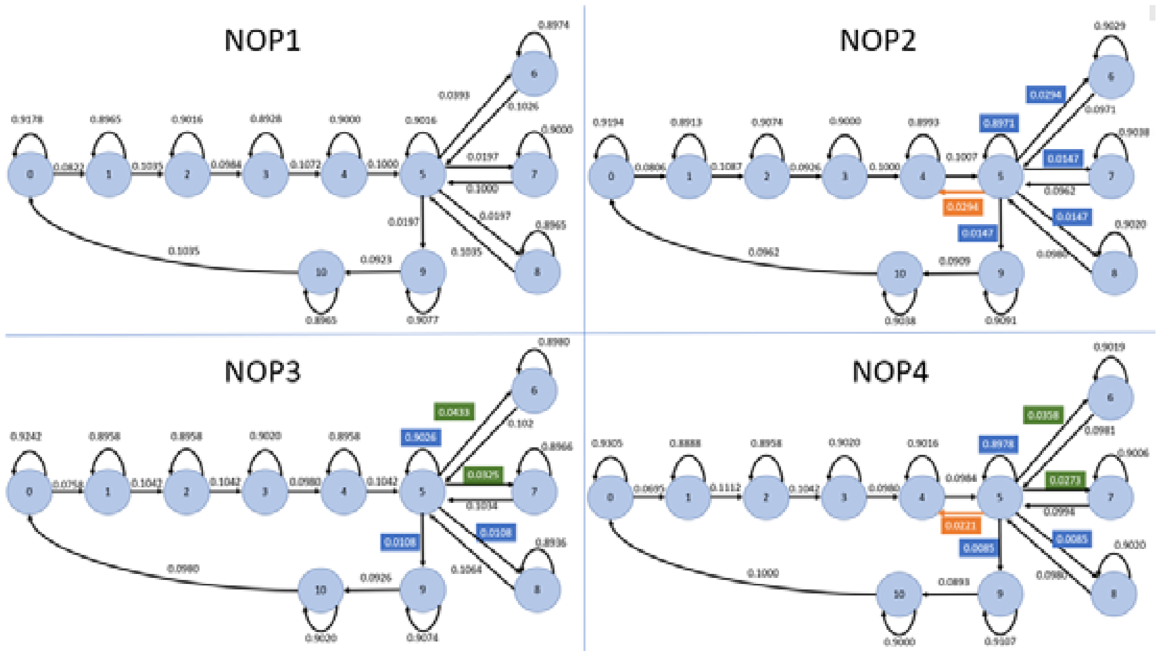
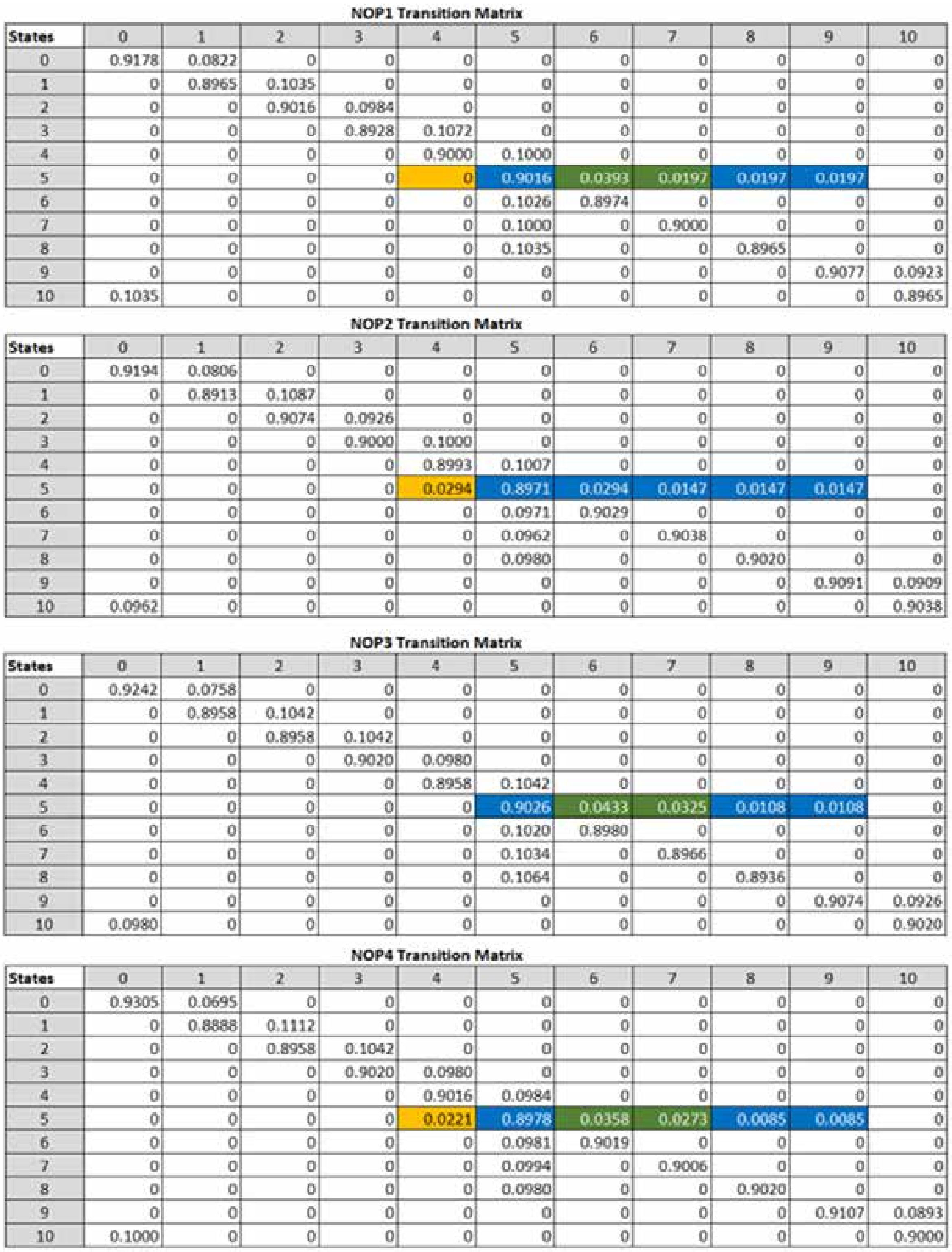


Figure 22. State transition diagrams for NOP1-NOP4



NOP1 Transition Matrix

States	0	1	2	3	4	5	6	7	8	9	10
0	0.9178	0.0822	0	0	0	0	0	0	0	0	0
1	0	0.8965	0.1035	0	0	0	0	0	0	0	0
2	0	0	0.9016	0.0984	0	0	0	0	0	0	0
3	0	0	0	0.8928	0.1072	0	0	0	0	0	0
4	0	0	0	0	0.9000	0.1000	0	0	0	0	0
5	0	0	0	0	0	0.9016	0.0393	0.0197	0.0197	0.0197	0
6	0	0	0	0	0	0.1026	0.8974	0	0	0	0
7	0	0	0	0	0	0.1000	0	0.9000	0	0	0
8	0	0	0	0	0	0.1035	0	0	0.8965	0	0
9	0	0	0	0	0	0	0	0	0	0.9077	0.0923
10	0.1035	0	0	0	0	0	0	0	0	0	0.8965

NOP2 Transition Matrix

States	0	1	2	3	4	5	6	7	8	9	10
0	0.9194	0.0806	0	0	0	0	0	0	0	0	0
1	0	0.8913	0.1087	0	0	0	0	0	0	0	0
2	0	0	0.9074	0.0926	0	0	0	0	0	0	0
3	0	0	0	0.9000	0.1000	0	0	0	0	0	0
4	0	0	0	0	0.8993	0.1007	0	0	0	0	0
5	0	0	0	0	0.0294	0.8971	0.0294	0.0147	0.0147	0.0147	0
6	0	0	0	0	0	0.0971	0.9029	0	0	0	0
7	0	0	0	0	0	0.0962	0	0.9038	0	0	0
8	0	0	0	0	0	0.0980	0	0	0.9020	0	0
9	0	0	0	0	0	0	0	0	0	0.9091	0.0909
10	0.0962	0	0	0	0	0	0	0	0	0	0.9038

NOP3 Transition Matrix

States	0	1	2	3	4	5	6	7	8	9	10
0	0.9242	0.0758	0	0	0	0	0	0	0	0	0
1	0	0.8958	0.1042	0	0	0	0	0	0	0	0
2	0	0	0.8958	0.1042	0	0	0	0	0	0	0
3	0	0	0	0.9020	0.0980	0	0	0	0	0	0
4	0	0	0	0	0.8958	0.1042	0	0	0	0	0
5	0	0	0	0	0	0.9026	0.0433	0.0325	0.0108	0.0108	0
6	0	0	0	0	0	0.1020	0.8980	0	0	0	0
7	0	0	0	0	0	0.1034	0	0.8966	0	0	0
8	0	0	0	0	0	0.1064	0	0	0.8936	0	0
9	0	0	0	0	0	0	0	0	0	0.9074	0.0926
10	0.0980	0	0	0	0	0	0	0	0	0	0.9020

NOP4 Transition Matrix

States	0	1	2	3	4	5	6	7	8	9	10
0	0.9305	0.0695	0	0	0	0	0	0	0	0	0
1	0	0.8888	0.1112	0	0	0	0	0	0	0	0
2	0	0	0.8958	0.1042	0	0	0	0	0	0	0
3	0	0	0	0.9020	0.0980	0	0	0	0	0	0
4	0	0	0	0	0.9016	0.0984	0	0	0	0	0
5	0	0	0	0	0.0221	0.8978	0.0358	0.0273	0.0085	0.0085	0
6	0	0	0	0	0	0.0981	0.9019	0	0	0	0
7	0	0	0	0	0	0.0994	0	0.9006	0	0	0
8	0	0	0	0	0	0.0980	0	0	0.9020	0	0
9	0	0	0	0	0	0	0	0	0	0.9107	0.0893
10	0.1000	0	0	0	0	0	0	0	0	0	0.9000

Figure 23. Transition matrices for NOP1-NOP4

The difference between NOP1 and NOP2 (in Figures 22 and 23) shows the effect of steps 5a/5b. Step 5a starts with the components in the condition represented by state 5 (fill pump is OFF), and then turns the fill pump to ON, resulting in the model creating a transition from state 5 to state 4 in NOP2. This transition is not present in the NOP1 model and is highlighted in orange in Figures 22 and 23 (including when it occurs for NOP4). Increasing the number of transitions from state 5 to state 4 will also slightly affect the transition probabilities for all transitions leaving state 5 in NOP 2. All the transitions that are indirectly affected in this way are highlighted blue in Figures 22 and 23, including those in NOP3 and NOP4. Once step 5a is complete the system is left idle for 5 seconds, causing a transition from state 4 back to state 4, and when step 5b is performed (turning the pump back to OFF), a transition from state 4 to state 5 is generated. Since both of these transitions are already seen in NOP1, no new transitions are created, and since the ratio of 5 seconds of transitions from state 4 to state 4 followed by a single transition to state 5 is maintained in both the NOP1 and NOP2 captures, no effect on the transition probabilities leaving state 4 is predicted.

C. SIMULATED ATTACKS

In addition to defining the NOP and generating the training models, the sequence-based attacks must be simulated against ISCICL and their network traffic must be captured to generate observation models. Since ISCICL network communications does not include any commands to reposition components, sequence-based attacks must be simulated by a manual operator performing component manipulations that violate the normal operating procedure. These simulated attacks are organized by the anomaly type they intend to generate.

1. Invalid State Attacks

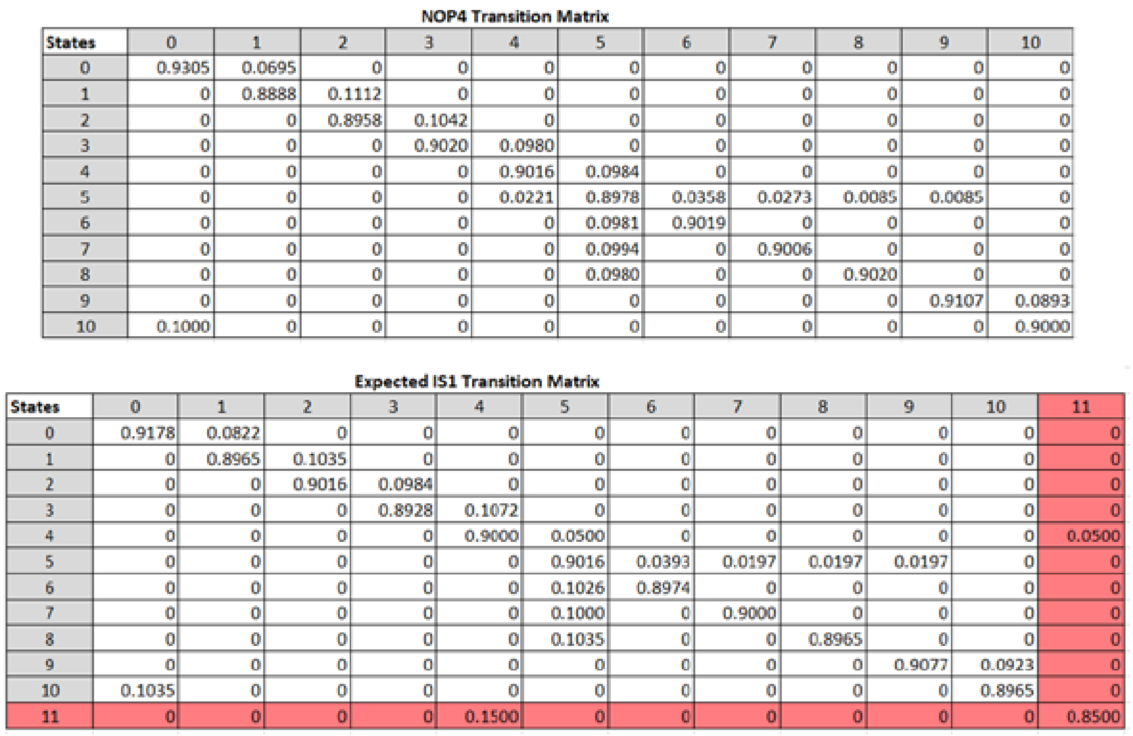
The objective of these attacks is to place the ICSICL system in a component configuration that does not exist in the NOP, and therefore test the ability to detect invalid state anomalies. There are two invalid state attacks, named IS1 and IS2. The corresponding traffic captures are referred to as IS1 and IS2 captures, and their models are referred to as IS1 and IS2 observation models, respectively. Table 10 defines the IS1 and IS2 attacks.

Table 10. Definition of invalid state traffic captures

Capture/Test Name	**Number of iterations of NOP**	**Special instructions**
IS1	3	During each iteration after step 4 is completed, place the HMI-override switch to OFF for five seconds, then return to ON before proceeding to the next step.
IS2	3	during each iteration after step 2 is completed, hold proximity sensor 2 for five seconds, then hold proximity sensor 3 for five seconds

Each IS attack will only consist of three iterations of the NOP. This was chosen to achieve a balance between having enough iterations to create consistency in the resulting model and limiting the number of total iterations for ease of manually operating the physical components when the attacks are conducted.

The IS1 attack will generate a new state in the IS1 observation model because the HMI-override switch is never moved to the OFF position during the NOP. This action will cause a unique value in the CIP command data section to be sent from the PLC to the HMI system during the communications cycle. Figures 24 and 25 illustrate the invalid state anomaly by showing the expected transition matrix and state diagram for the IS1 observation model alongside those of the NOP4 model. IS1 is expected to generate a new state, called state 11, when the HMI switch is repositioned. The effect of this new state on the transition matrix and state diagram is highlighted in red in Figures 24 and 25.



NOP4 Transition Matrix

States	0	1	2	3	4	5	6	7	8	9	10
0	0.9305	0.0695	0	0	0	0	0	0	0	0	0
1	0	0.8888	0.1112	0	0	0	0	0	0	0	0
2	0	0	0.8958	0.1042	0	0	0	0	0	0	0
3	0	0	0	0.9020	0.0980	0	0	0	0	0	0
4	0	0	0	0	0.9016	0.0984	0	0	0	0	0
5	0	0	0	0	0.0221	0.8978	0.0358	0.0273	0.0085	0.0085	0
6	0	0	0	0	0	0.0981	0.9019	0	0	0	0
7	0	0	0	0	0	0.0994	0	0.9006	0	0	0
8	0	0	0	0	0	0.0980	0	0	0.9020	0	0
9	0	0	0	0	0	0	0	0	0	0.9107	0.0893
10	0.1000	0	0	0	0	0	0	0	0	0	0.9000

Expected IS1 Transition Matrix

States	0	1	2	3	4	5	6	7	8	9	10	11
0	0.9178	0.0822	0	0	0	0	0	0	0	0	0	0
1	0	0.8965	0.1035	0	0	0	0	0	0	0	0	0
2	0	0	0.9016	0.0984	0	0	0	0	0	0	0	0
3	0	0	0	0.8928	0.1072	0	0	0	0	0	0	0
4	0	0	0	0	0.9000	0.0500	0	0	0	0	0	0.0500
5	0	0	0	0	0	0.9016	0.0393	0.0197	0.0197	0.0197	0	0
6	0	0	0	0	0	0.1026	0.8974	0	0	0	0	0
7	0	0	0	0	0	0.1000	0	0.9000	0	0	0	0
8	0	0	0	0	0	0.1035	0	0	0.8965	0	0	0
9	0	0	0	0	0	0	0	0	0	0.9077	0.0923	0
10	0.1035	0	0	0	0	0	0	0	0	0	0.8965	0
11	0	0	0	0	0.1500	0	0	0	0	0	0	0.8500

Figure 24. Expected transition matrix for IS1 attack

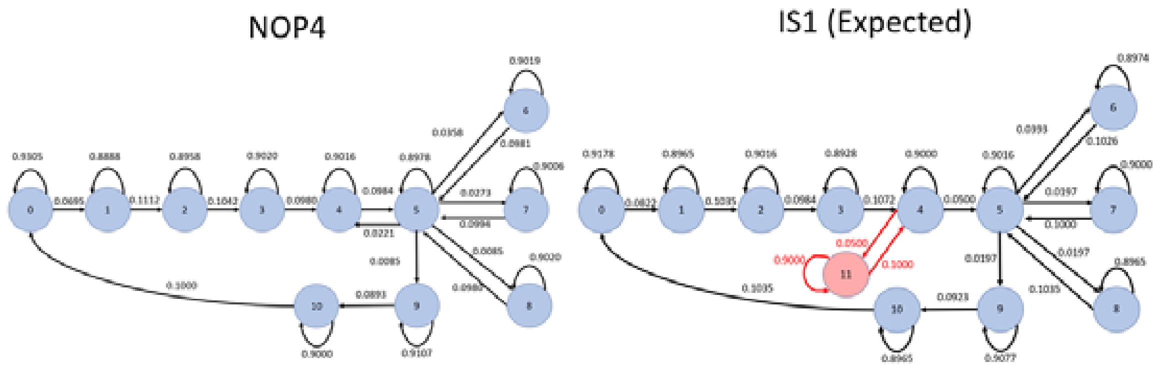
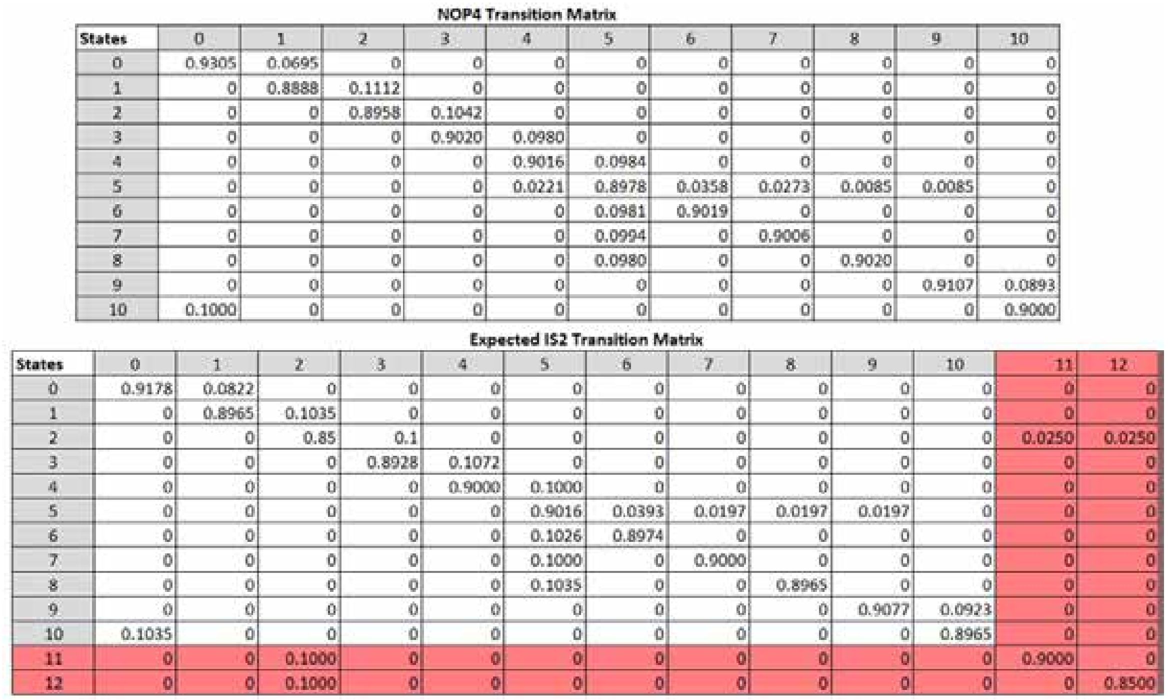


Figure 25. Expected state diagram for IS1 attack

The IS2 attack will generate two new states in the IS2 observation model (state 11 and state 12) because the proximity sensors are only activated when the AFTL pump is OFF, valve is SHUT, and high-level alarm is LIT. Since the DIOL status is communicated in the same twelve bytes as the AFTL status, manipulating the DIOL when the AFTL is in any other condition will result in a unique value in the command data section. State 11 represents the activation of proximity sensor #2 while the fill pump is ON, drain valve is SHUT and low-level alarm is LIT (represented as state #2). State 12 represents the activation of proximity sensor #3 in these conditions. Figures 26 and 27 show the transition matrix and state diagram of the expected model of the IS2 attack together with the transition matrix and state diagram of the NOP4 model. States 11 and 12 are highlighted to show the expected invalid state anomalies.



NOP4 Transition Matrix

States	0	1	2	3	4	5	6	7	8	9	10
0	0.9305	0.0695	0	0	0	0	0	0	0	0	0
1	0	0.8888	0.1112	0	0	0	0	0	0	0	0
2	0	0	0.8958	0.1042	0	0	0	0	0	0	0
3	0	0	0	0.9020	0.0980	0	0	0	0	0	0
4	0	0	0	0	0.9016	0.0984	0	0	0	0	0
5	0	0	0	0	0.0221	0.8978	0.0358	0.0273	0.0085	0.0085	0
6	0	0	0	0	0	0.0981	0.9019	0	0	0	0
7	0	0	0	0	0	0.0994	0	0.9006	0	0	0
8	0	0	0	0	0	0.0980	0	0	0.9020	0	0
9	0	0	0	0	0	0	0	0	0	0.9107	0.0893
10	0.1000	0	0	0	0	0	0	0	0	0	0.9000

Expected IS2 Transition Matrix

States	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.9178	0.0822	0	0	0	0	0	0	0	0	0	0	0
1	0	0.8965	0.1035	0	0	0	0	0	0	0	0	0	0
2	0	0	0.85	0.1	0	0	0	0	0	0	0	0.0250	0.0250
3	0	0	0	0.8928	0.1072	0	0	0	0	0	0	0	0
4	0	0	0	0	0.9000	0.1000	0	0	0	0	0	0	0
5	0	0	0	0	0	0.9016	0.0393	0.0197	0.0197	0.0197	0	0	0
6	0	0	0	0	0	0.1026	0.8974	0	0	0	0	0	0
7	0	0	0	0	0	0.1000	0	0.9000	0	0	0	0	0
8	0	0	0	0	0	0.1035	0	0	0.8965	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0.9077	0.0923	0	0
10	0.1035	0	0	0	0	0	0	0	0	0	0.8965	0	0
11	0	0	0.1000	0	0	0	0	0	0	0	0	0.9000	0
12	0	0	0.1000	0	0	0	0	0	0	0	0	0	0.8500

Figure 26. Expected transition matrix for IS2 attack

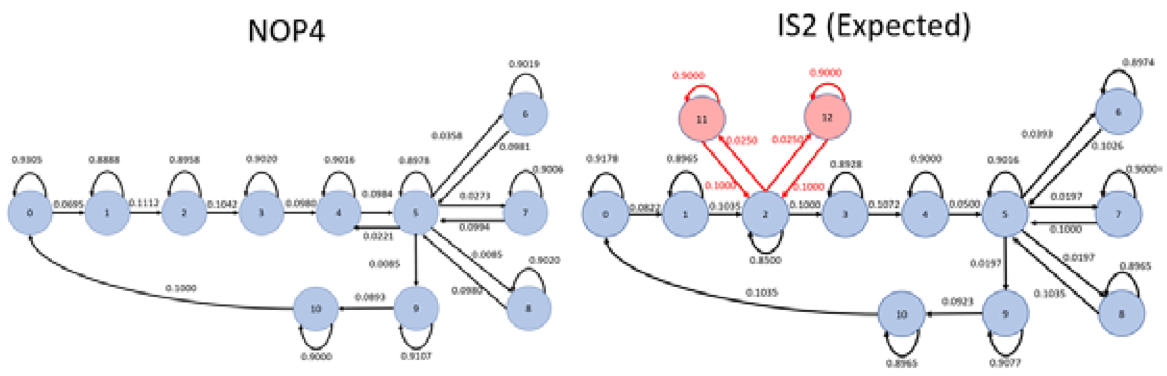


Figure 27. Expected state transition diagram for IS2 attack

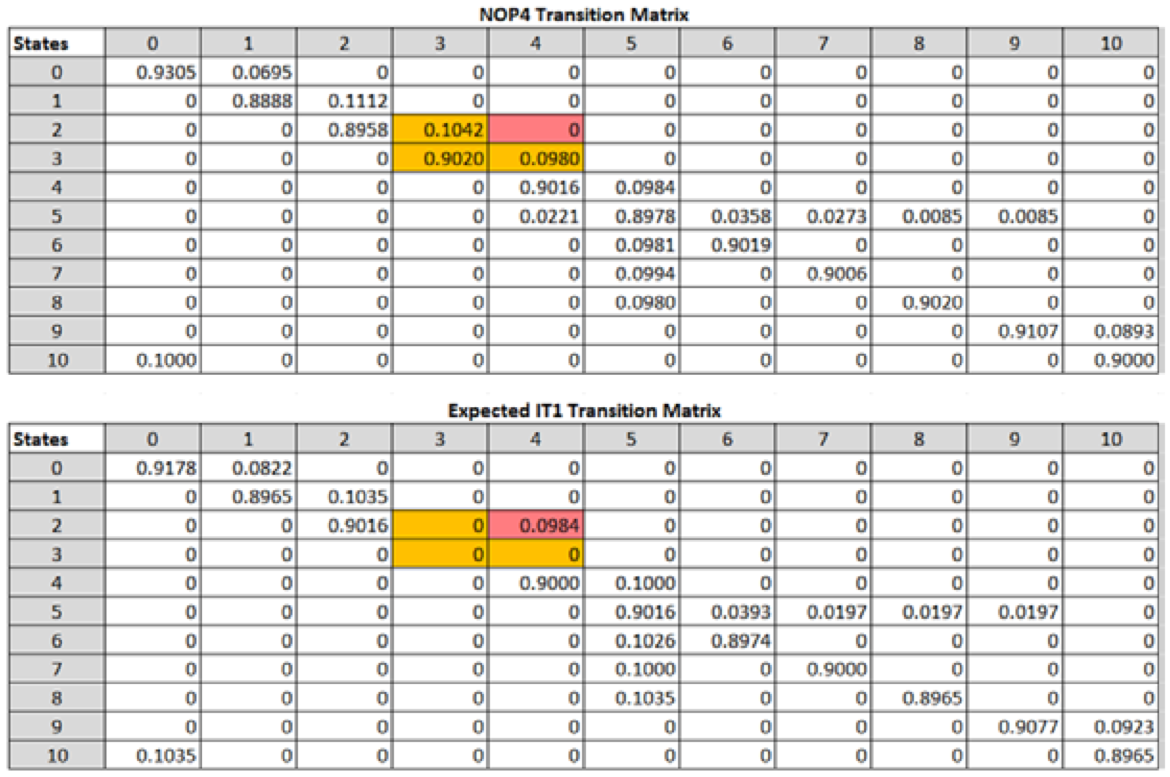
2. Invalid Transition Attacks

There are two invalid transition attacks, named IT1 and IT2. These attacks are intended to create a new transition between two valid states of the NOP4 training model, testing the ability to detect invalid transition anomalies. The corresponding traffic captures are referred to as IT1 and IT2 captures, and their models are referred to as IT1 and 2 observation models, respectively. As with the IS attacks, each IT attack will only consist of three iterations of the NOP. The reasoning behind conducting three iteration is the same as presented for the IS attacks. Table 11 defines the IT1 and IT2 attacks.

Table 11. Definition of invalid transition traffic captures

Capture/Test Name	**Number of iterations of NOP**	**Special instructions**
IT1	3	During each iteration step 3 is skipped, and the tank level control knob is quickly turned to the fully clockwise position.
IT2	3	During each iteration steps 1 and 2 are performed at the same time.

The IT1 attack will generate a new transition between states 2 and 4 of the NOP4 model (see Figure 7) by skipping the state where neither of the tank-level alarms is lit (state 3), causing the IT1 observation model to contain the transition from the AFTL component status with the low-level alarm LIT directly to the AFTL component status with the high-level alarm LIT, which is not in the training model. Figures 28 and 29 show the transition matrix and state diagram of the expected model of the IT1 attack along with the transition matrix and state diagram of NOP4 model. The new transition from state 2 to state 4 is highlighted in red and is expected to cause a transition anomaly. Additionally, since state 3 is entirely skipped, the expected model does not contain any transitions to or from state 3. The transitions that are affected by skipping state 3 are highlighted in orange, with the new transition from state 2 to state 4 highlighted in red.



NOP4 Transition Matrix

States	0	1	2	3	4	5	6	7	8	9	10
0	0.9305	0.0695	0	0	0	0	0	0	0	0	0
1	0	0.8888	0.1112	0	0	0	0	0	0	0	0
2	0	0	0.8958	0.1042	0	0	0	0	0	0	0
3	0	0	0	0.9020	0.0980	0	0	0	0	0	0
4	0	0	0	0	0.9016	0.0984	0	0	0	0	0
5	0	0	0	0	0.0221	0.8978	0.0358	0.0273	0.0085	0.0085	0
6	0	0	0	0	0	0.0981	0.9019	0	0	0	0
7	0	0	0	0	0	0.0994	0	0.9006	0	0	0
8	0	0	0	0	0	0.0980	0	0	0.9020	0	0
9	0	0	0	0	0	0	0	0	0	0.9107	0.0893
10	0.1000	0	0	0	0	0	0	0	0	0	0.9000

Expected IT1 Transition Matrix

States	0	1	2	3	4	5	6	7	8	9	10
0	0.9178	0.0822	0	0	0	0	0	0	0	0	0
1	0	0.8965	0.1035	0	0	0	0	0	0	0	0
2	0	0	0.9016	0	0.0984	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0.9000	0.1000	0	0	0	0	0
5	0	0	0	0	0	0.9016	0.0393	0.0197	0.0197	0.0197	0
6	0	0	0	0	0	0.1026	0.8974	0	0	0	0
7	0	0	0	0	0	0.1000	0	0.9000	0	0	0
8	0	0	0	0	0	0.1035	0	0	0.8965	0	0
9	0	0	0	0	0	0	0	0	0	0.9077	0.0923
10	0.1035	0	0	0	0	0	0	0	0	0	0.8965

Figure 28. Expected transition matrix for IT1 attack

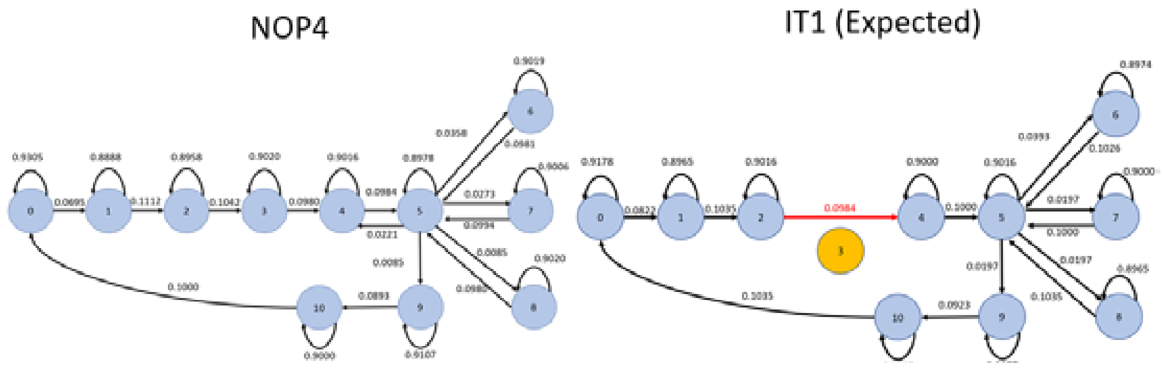
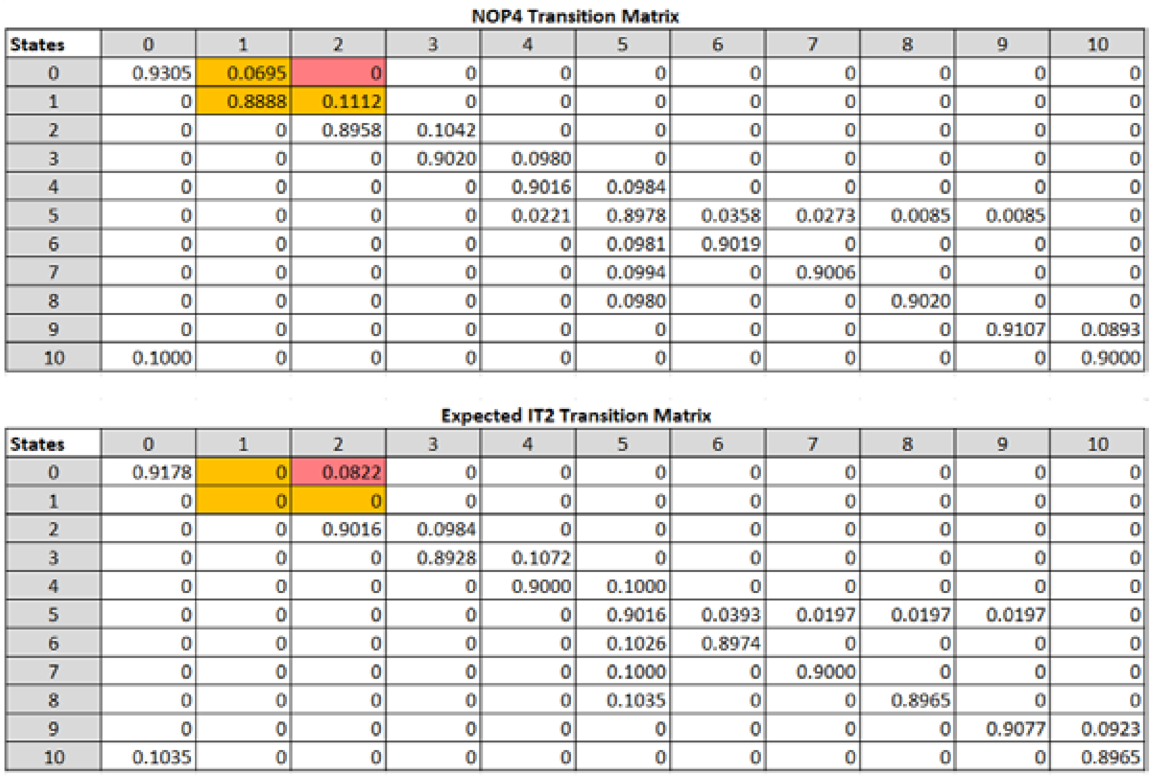


Figure 29. Expected state diagram for IT1 attack

The IT2 attack will generate a new transition between states 0 and 2 of the NOP4 model (see Table 7) by skipping the state where the drain valve is SHUT but the fill pump is OFF (state 1), causing the IT2 observation model to transition from state 0 directly to state 2. Figures 30 and 31 show the transition matrices and state diagrams of the expected model of the IT2 attack and the NOP4 model. The new transition from state 0 to state 2 is highlighted in red and is expected to cause a transition anomaly. Additionally, since state 1 is entirely skipped, the expected model does not contain any transitions to or from state 1. The transitions that are affected by skipping state 1 are highlighted in orange, and the new transition from state 0 to state 2 highlighted in red.



NOP4 Transition Matrix

States	0	1	2	3	4	5	6	7	8	9	10
0	0.9305	0.0695	0	0	0	0	0	0	0	0	0
1	0	0.8888	0.1112	0	0	0	0	0	0	0	0
2	0	0	0.8958	0.1042	0	0	0	0	0	0	0
3	0	0	0	0.9020	0.0980	0	0	0	0	0	0
4	0	0	0	0	0.9016	0.0984	0	0	0	0	0
5	0	0	0	0	0.0221	0.8978	0.0358	0.0273	0.0085	0.0085	0
6	0	0	0	0	0	0.0981	0.9019	0	0	0	0
7	0	0	0	0	0	0.0994	0	0.9006	0	0	0
8	0	0	0	0	0	0.0980	0	0	0.9020	0	0
9	0	0	0	0	0	0	0	0	0	0.9107	0.0893
10	0.1000	0	0	0	0	0	0	0	0	0	0.9000

Expected IT2 Transition Matrix

States	0	1	2	3	4	5	6	7	8	9	10
0	0.9178	0	0.0822	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0.9016	0.0984	0	0	0	0	0	0	0
3	0	0	0	0.8928	0.1072	0	0	0	0	0	0
4	0	0	0	0	0.9000	0.1000	0	0	0	0	0
5	0	0	0	0	0	0.9016	0.0393	0.0197	0.0197	0.0197	0
6	0	0	0	0	0	0.1026	0.8974	0	0	0	0
7	0	0	0	0	0	0.1000	0	0.9000	0	0	0
8	0	0	0	0	0	0.1035	0	0	0.8965	0	0
9	0	0	0	0	0	0	0	0	0	0.9077	0.0923
10	0.1035	0	0	0	0	0	0	0	0	0	0.8965

Figure 30. Expected transition matrix for IT2 attack

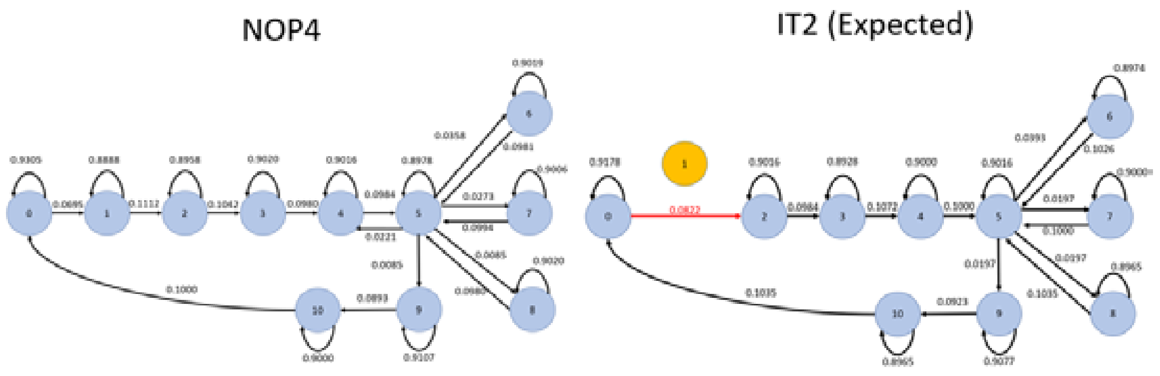


Figure 31. Expected state diagram for IT2 attack

57

3. Invalid Transition Probability Attacks

Each of these attacks is expected to create an observation model that contains transition probabilities that differ from those found in the NOP4 training model by more than Σ. The initial Σ is 0.015 as described in section C.1 of this chapter, and Σ will remain at that value for all subsequent tests. Additionally, localized transition anomalies at various points within the traffic capture are introduced to compare the performances of the proposed detection methods. These generated traffic captures are referred to as IP1 through IP9 captures, and their models are referred to IP1 through IP9 observation models, respectively. Table 12 defines the IP1-IP9 attacks.

Table 12. Definition of invalid probability traffic captures

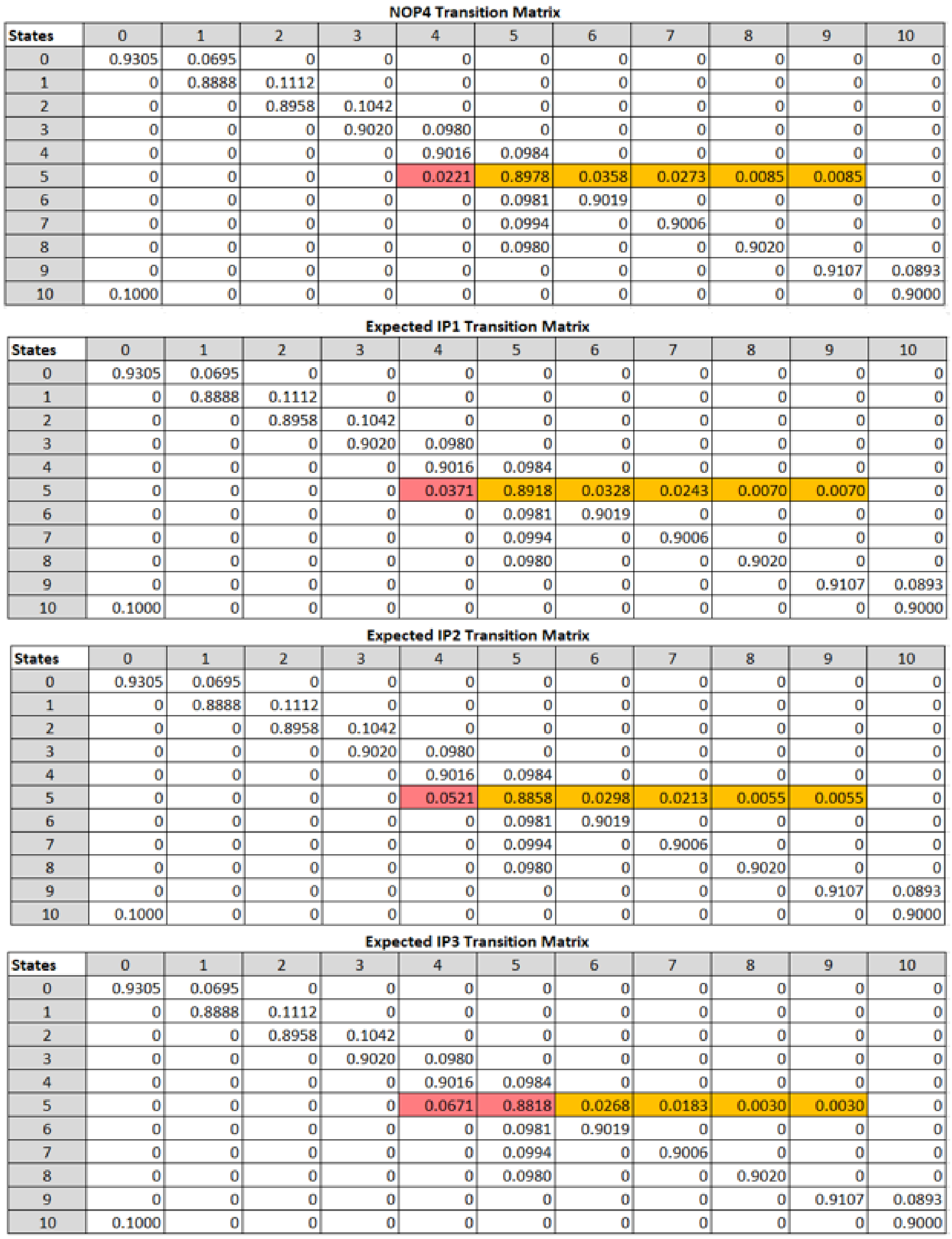
Capture/Test Name	**Number of iterations of NOP**	**Special instructions**
IP1	3	During each iteration steps 5a/5b are repeated three times
IP2	3	During each iteration steps 5a/5b are repeated five times
IP3	3	During each iteration steps 5a/5b are repeated ten times
IP4	5	During the first iteration steps 5a/5b are repeated five times. All other iterations are completed as normal
IP5	5	During the first iteration steps 5a/5b are repeated ten times. All other iterations are completed as normal
IP6	5	During the third iteration steps 5a/5b are repeated five times. All other iterations are completed as normal
IP7	5	During the third iteration steps 5a/5b are repeated ten times. All other iterations are completed as normal
IP8	5	During the last iteration steps 5a/5b are repeated five times. All other iterations are completed as normal
IP9	5	During the last iteration steps 5a/5b are repeated ten times. All other iterations are completed as normal

a. IP1 through IP3

The IP1 through IP3 attacks consist of three iteration of the NOP for the same reasons as discussed for IS and IT attacks. The actions in captures IP1 through IP3 violate the NOP during each iteration by the cycling the fill pump above the number of times that the NOP allow. The amount of cycling is designed to simulate attacks that require different levels of difficult to detect: high (IP1), medium (IP2), and low (IP3). IP1 does this cycling only one extra time, and while it is expected that this action will trigger a probability anomaly, the transition probabilities in IP1 will have the smallest deviation from the corresponding transition probabilities in NOP4. IP2 and IP3 increase the number of cycling of the drain pump to five and ten respectively, therefore increasing the magnitude of the difference between their transition probabilities and those in NOP4.

Figures 32 and 33 show the transition matrixes and state diagrams for the IP1–IP3 attacks and the NOP4 model. Because each of these attacks increases the number of times state 5 transitions to state 4, the transition probability from state 5 to state 4 is expected to show the largest change. However, it is important to note that all of the transition probabilities from state 5 will be affected. It is expected that the invalid probability anomaly is triggered by the comparison of the IP attack's probability value for the transition from state 5 to state 4 to NOP4's probability value for the transition from state 5 to state 4. However, for the IP3 attack, the amount that this transition's probability is changed could also cause other transition's probabilities from state 5 to differ from their corresponding probability in NOP4 by more than Σ, triggering multiple transition anomalies. The expected transition probabilities that will trigger the transition anomaly are highlighted in red.

The attacks in IP1-IP3 are included in each iteration of the NOP, and the statistical effect of the attack on transition probabilities is approximately the same for each of the iterations within the capture. Therefore, all five probability anomaly detection methods are expected to correctly identify the probability anomaly, and since each iteration during the attack is the same, all probability anomaly detection methods are expected to have approximately the same transition matrix. Therefore, each IP transition matrix and state diagram displayed in Figures 32 and 33 represent the expected performance of each anomaly detection method. The specific transition probabilities that are expected to cause probability anomalies are highlighted in red, while the probabilities that are expected to be affected, but not enough to cause an anomaly are highlighted in orange.



NOP4 Transition Matrix

States	0	1	2	3	4	5	6	7	8	9	10
0	0.9305	0.0695	0	0	0	0	0	0	0	0	0
1	0	0.8888	0.1112	0	0	0	0	0	0	0	0
2	0	0	0.8958	0.1042	0	0	0	0	0	0	0
3	0	0	0	0.9020	0.0980	0	0	0	0	0	0
4	0	0	0	0	0.9016	0.0984	0	0	0	0	0
5	0	0	0	0	0.0221	0.8978	0.0358	0.0273	0.0085	0.0085	0
6	0	0	0	0	0	0.0981	0.9019	0	0	0	0
7	0	0	0	0	0	0.0994	0	0.9006	0	0	0
8	0	0	0	0	0	0.0980	0	0	0.9020	0	0
9	0	0	0	0	0	0	0	0	0	0.9107	0.0893
10	0.1000	0	0	0	0	0	0	0	0	0	0.9000

Expected IP1 Transition Matrix

States	0	1	2	3	4	5	6	7	8	9	10
0	0.9305	0.0695	0	0	0	0	0	0	0	0	0
1	0	0.8888	0.1112	0	0	0	0	0	0	0	0
2	0	0	0.8958	0.1042	0	0	0	0	0	0	0
3	0	0	0	0.9020	0.0980	0	0	0	0	0	0
4	0	0	0	0	0.9016	0.0984	0	0	0	0	0
5	0	0	0	0	0.0371	0.8918	0.0328	0.0243	0.0070	0.0070	0
6	0	0	0	0	0	0.0981	0.9019	0	0	0	0
7	0	0	0	0	0	0.0994	0	0.9006	0	0	0
8	0	0	0	0	0	0.0980	0	0	0.9020	0	0
9	0	0	0	0	0	0	0	0	0	0.9107	0.0893
10	0.1000	0	0	0	0	0	0	0	0	0	0.9000

Expected IP2 Transition Matrix

States	0	1	2	3	4	5	6	7	8	9	10
0	0.9305	0.0695	0	0	0	0	0	0	0	0	0
1	0	0.8888	0.1112	0	0	0	0	0	0	0	0
2	0	0	0.8958	0.1042	0	0	0	0	0	0	0
3	0	0	0	0.9020	0.0980	0	0	0	0	0	0
4	0	0	0	0	0.9016	0.0984	0	0	0	0	0
5	0	0	0	0	0.0521	0.8858	0.0298	0.0213	0.0055	0.0055	0
6	0	0	0	0	0	0.0981	0.9019	0	0	0	0
7	0	0	0	0	0	0.0994	0	0.9006	0	0	0
8	0	0	0	0	0	0.0980	0	0	0.9020	0	0
9	0	0	0	0	0	0	0	0	0	0.9107	0.0893
10	0.1000	0	0	0	0	0	0	0	0	0	0.9000

Expected IP3 Transition Matrix

States	0	1	2	3	4	5	6	7	8	9	10
0	0.9305	0.0695	0	0	0	0	0	0	0	0	0
1	0	0.8888	0.1112	0	0	0	0	0	0	0	0
2	0	0	0.8958	0.1042	0	0	0	0	0	0	0
3	0	0	0	0.9020	0.0980	0	0	0	0	0	0
4	0	0	0	0	0.9016	0.0984	0	0	0	0	0
5	0	0	0	0	0.0671	0.8818	0.0268	0.0183	0.0030	0.0030	0
6	0	0	0	0	0	0.0981	0.9019	0	0	0	0
7	0	0	0	0	0	0.0994	0	0.9006	0	0	0
8	0	0	0	0	0	0.0980	0	0	0.9020	0	0
9	0	0	0	0	0	0	0	0	0	0.9107	0.0893
10	0.1000	0	0	0	0	0	0	0	0	0	0.9000

Figure 32. Expected transition matrixes for IP1-IP3 attacks

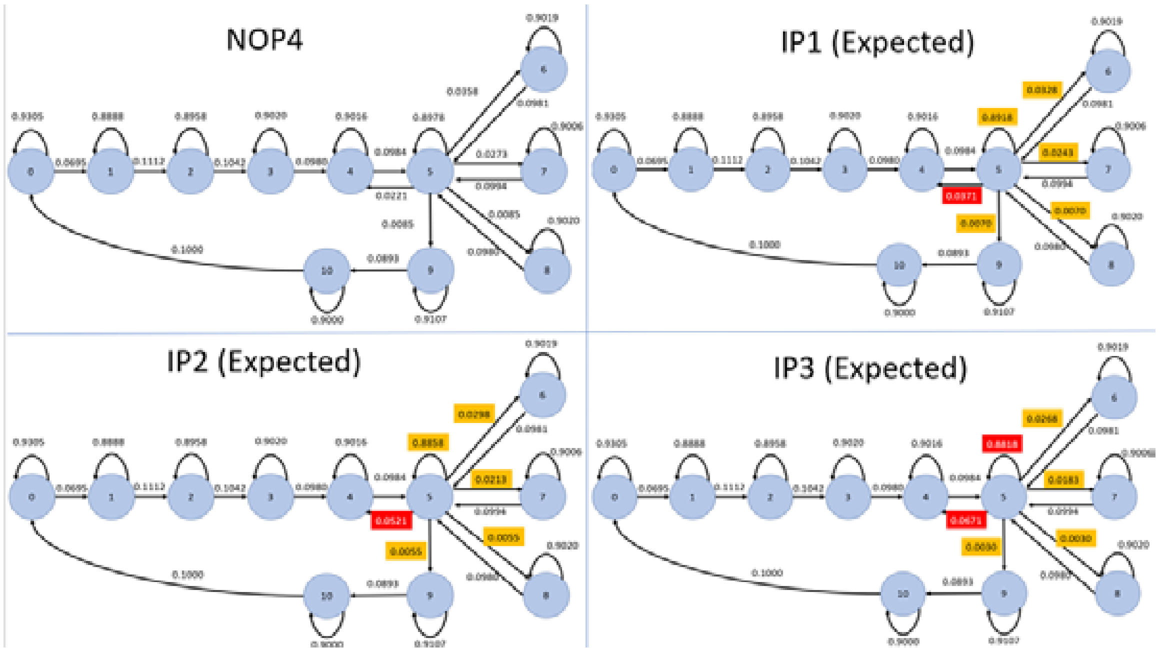


Figure 33. Expected state diagram for IP1-IP3 attacks

b. IP4 through IP9 attacks

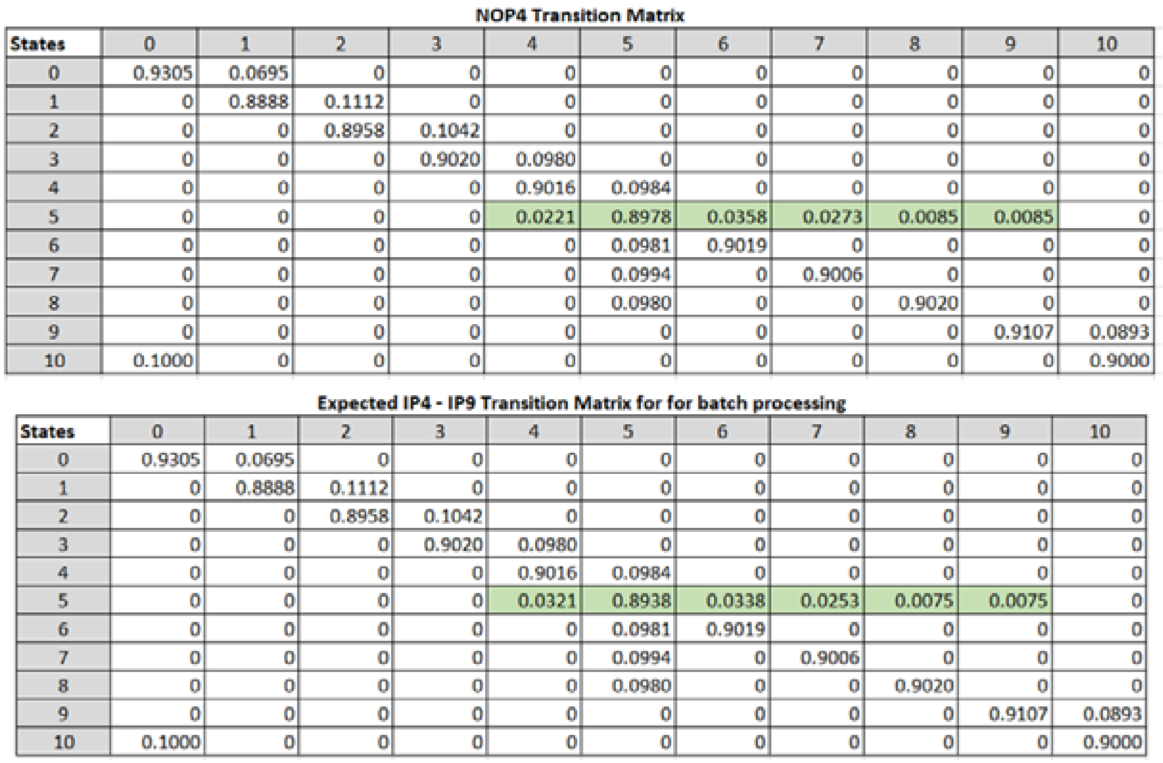
For the IP4 - IP9 attacks, only the actions in the iteration that cycles the fill pump above the number of times that the NOP allows violate the NOP. The number of iterations of the NOP used in these attacks is increased to five to provide more iterations that do not contain a violation of the NOP, therefore decreasing the statistical effect the single attack containing iteration has on the overall model. This makes localized anomaly detection more difficult and will give more insight to the differences between the probability anomaly detection methods. Specifically, batch processing is expected to fail to detect all of these attacks, while stream processing is expected to fail to detect attacks IP6 – IP9. Batch and stream processing methods are expected to have difficulty detecting localized anomalies, while repeat processing, sliding window, and alpha filter methods are expected to successfully detect them. The IP4 – IP9 attacks are designed to test this hypothesis.

IP4 and IP5 place the localized anomaly in the first iteration of the capture. In IP6 and IP7, the localized anomaly occurs in the 3rd iteration of the capture, and in IP8 and IP9 the localized namely occurs in the 5th and final iteration of the capture. These differences allow tests to determine if the performance (successful detection) of the anomaly detection methods is dependent on where the localized anomaly is within the capture. Additionally, the localized anomaly in IP4, IP6, and IP8 consists of five cycles of performing steps 5a/5b while IP5, IP7, IP9 consists of ten cycles of steps 5a/5b. This provides two levels of magnitude of the localized anomaly and is included to determine if some anomaly detection methods can find large localized anomalies but not smaller ones. However, the expectation is that if an anomaly detection method is expected to detect a localized anomaly, the magnitude of that anomaly does not matter. Therefore, the expected transition matrixes and state diagrams shown in this section do not differentiate between larger and smaller versions of localized anomalies.

(1) IP4 through IP9: Batch Processing

Batch processing makes a single observation model from every packet in the observation capture, and then compares the observation model to the training model. Since the anomaly in IP4-IP9 only occurs during one of five iterations, the statistical effect on the transition probabilities will likely not be of large enough magnitude for the batch processing method to detect.

Figures 34 and 35 show the transition matrix and state diagram of the IP4-IP9 attacks when processed with the batch processing method. The transition probabilities from state 5 are highlighted green to show that while it did change from the probability found in NOP4, the magnitude of the expected change is less than Σ; therefore, no anomaly is detected.



NOP4 Transition Matrix

States	0	1	2	3	4	5	6	7	8	9	10
0	0.9305	0.0695	0	0	0	0	0	0	0	0	0
1	0	0.8888	0.1112	0	0	0	0	0	0	0	0
2	0	0	0.8958	0.1042	0	0	0	0	0	0	0
3	0	0	0	0.9020	0.0980	0	0	0	0	0	0
4	0	0	0	0	0.9016	0.0984	0	0	0	0	0
5	0	0	0	0	0.0221	0.8978	0.0358	0.0273	0.0085	0.0085	0
6	0	0	0	0	0	0.0981	0.9019	0	0	0	0
7	0	0	0	0	0	0.0994	0	0.9006	0	0	0
8	0	0	0	0	0	0.0980	0	0	0.9020	0	0
9	0	0	0	0	0	0	0	0	0	0.9107	0.0893
10	0.1000	0	0	0	0	0	0	0	0	0	0.9000

Expected IP4 - IP9 Transition Matrix for for batch processing

States	0	1	2	3	4	5	6	7	8	9	10
0	0.9305	0.0695	0	0	0	0	0	0	0	0	0
1	0	0.8888	0.1112	0	0	0	0	0	0	0	0
2	0	0	0.8958	0.1042	0	0	0	0	0	0	0
3	0	0	0	0.9020	0.0980	0	0	0	0	0	0
4	0	0	0	0	0.9016	0.0984	0	0	0	0	0
5	0	0	0	0	0.0321	0.8938	0.0338	0.0253	0.0075	0.0075	0
6	0	0	0	0	0	0.0981	0.9019	0	0	0	0
7	0	0	0	0	0	0.0994	0	0.9006	0	0	0
8	0	0	0	0	0	0.0980	0	0	0.9020	0	0
9	0	0	0	0	0	0	0	0	0	0.9107	0.0893
10	0.1000	0	0	0	0	0	0	0	0	0	0.9000

Figure 34. Expected transition matrix for IP4-IP9 attacks with batch processing method

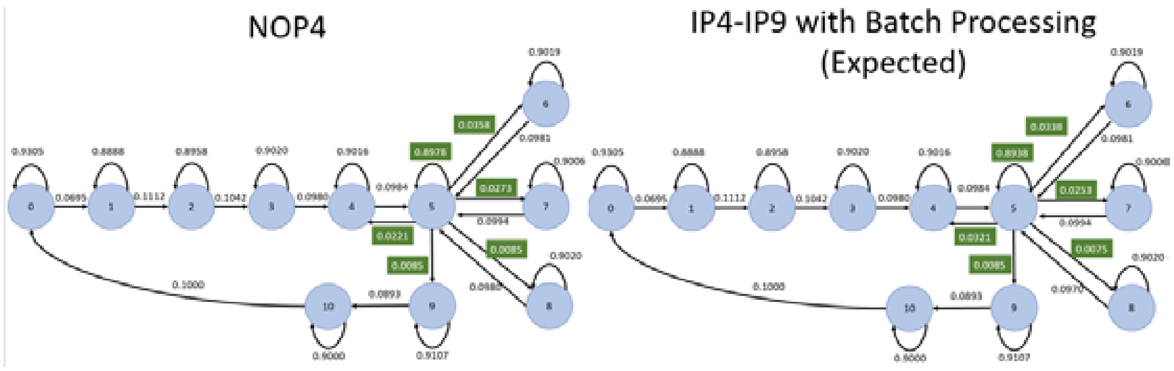
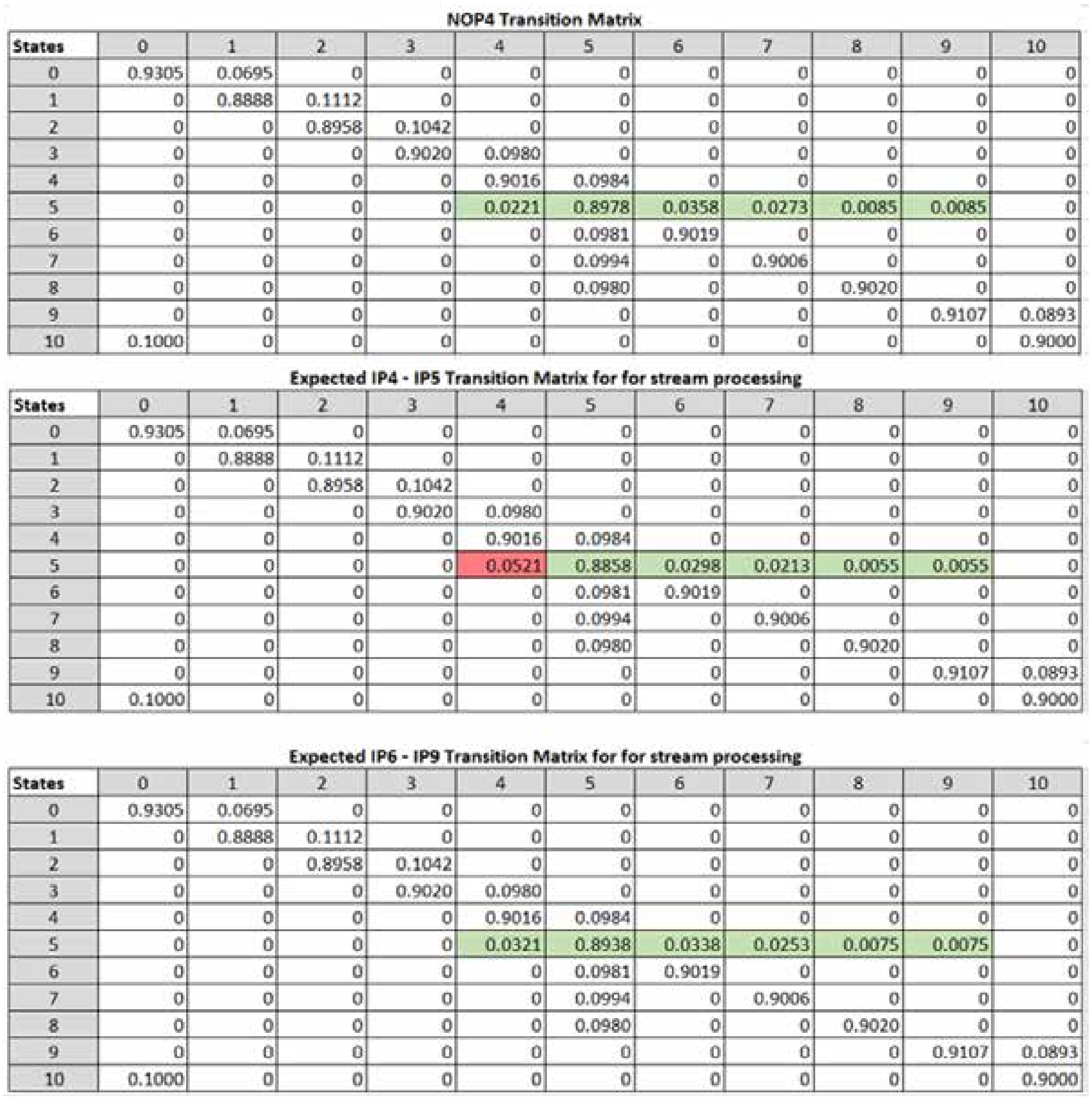


Figure 35. Expected state diagram for IP4-IP9 attacks with batch processing method

(2) IP4 through IP9: Stream Processing

Stream processing checks for anomalies several times as a single observation model is getting built, and therefore can detect localized anomalies if they occur early in the observation capture. However, if the localized anomalies occur late in the observation capture the statistical effect on the transition probabilities will likely not be of large enough magnitude for the stream processing method to detect. Therefore, stream processing is expected to detect the anomaly in IP4 and IP5, but not in IP6-IP9.

Figures 36 and 37 show the transition matrix and state diagram for the IP4-IP5 attacks and IP6-IP9 attacks when processed with the stream processing method. The expected successful detection of the transition anomaly early in the capture (IP5-IP5) is shown in red, indicating the transition probability that differs with the corresponding NOP4 transition probability by more than Σ. The other transition probabilities leaving state 5 are highlighted in green to show that while they change, their expected change is less than Σ and should not cause an anomaly. The expected failure to detect the anomaly later in the capture (IP6-IP9) is shown in green, showing the transition probabilities that fail to differ from the corresponding NOP4 transition probabilities by more than Σ. Additionally, since stream processing involves comparing many different observation models, only the expected observation model that triggers the anomaly is shown. All other generations of the observation model would be redundant since once a single generation detects an anomaly the entire observation capture is considered to contain an anomaly.



NOP4 Transition Matrix

States	0	1	2	3	4	5	6	7	8	9	10
0	0.9305	0.0695	0	0	0	0	0	0	0	0	0
1	0	0.8888	0.1112	0	0	0	0	0	0	0	0
2	0	0	0.8958	0.1042	0	0	0	0	0	0	0
3	0	0	0	0.9020	0.0980	0	0	0	0	0	0
4	0	0	0	0	0.9016	0.0984	0	0	0	0	0
5	0	0	0	0	0.0221	0.8978	0.0358	0.0273	0.0085	0.0085	0
6	0	0	0	0	0	0.0981	0.9019	0	0	0	0
7	0	0	0	0	0	0.0994	0	0.9006	0	0	0
8	0	0	0	0	0	0.0980	0	0	0.9020	0	0
9	0	0	0	0	0	0	0	0	0	0.9107	0.0893
10	0.1000	0	0	0	0	0	0	0	0	0	0.9000

Expected IP4 - IP5 Transition Matrix for for stream processing

States	0	1	2	3	4	5	6	7	8	9	10
0	0.9305	0.0695	0	0	0	0	0	0	0	0	0
1	0	0.8888	0.1112	0	0	0	0	0	0	0	0
2	0	0	0.8958	0.1042	0	0	0	0	0	0	0
3	0	0	0	0.9020	0.0980	0	0	0	0	0	0
4	0	0	0	0	0.9016	0.0984	0	0	0	0	0
5	0	0	0	0	0.0521	0.8858	0.0298	0.0213	0.0055	0.0055	0
6	0	0	0	0	0	0.0981	0.9019	0	0	0	0
7	0	0	0	0	0	0.0994	0	0.9006	0	0	0
8	0	0	0	0	0	0.0980	0	0	0.9020	0	0
9	0	0	0	0	0	0	0	0	0	0.9107	0.0893
10	0.1000	0	0	0	0	0	0	0	0	0	0.9000

Expected IP6 - IP9 Transition Matrix for for stream processing

States	0	1	2	3	4	5	6	7	8	9	10
0	0.9305	0.0695	0	0	0	0	0	0	0	0	0
1	0	0.8888	0.1112	0	0	0	0	0	0	0	0
2	0	0	0.8958	0.1042	0	0	0	0	0	0	0
3	0	0	0	0.9020	0.0980	0	0	0	0	0	0
4	0	0	0	0	0.9016	0.0984	0	0	0	0	0
5	0	0	0	0	0.0321	0.8938	0.0338	0.0253	0.0075	0.0075	0
6	0	0	0	0	0	0.0981	0.9019	0	0	0	0
7	0	0	0	0	0	0.0994	0	0.9006	0	0	0
8	0	0	0	0	0	0.0980	0	0	0.9020	0	0
9	0	0	0	0	0	0	0	0	0	0.9107	0.0893
10	0.1000	0	0	0	0	0	0	0	0	0	0.9000

Figure 36. Expected transition matrixes for IP4-IP6 attacks with stream processing method

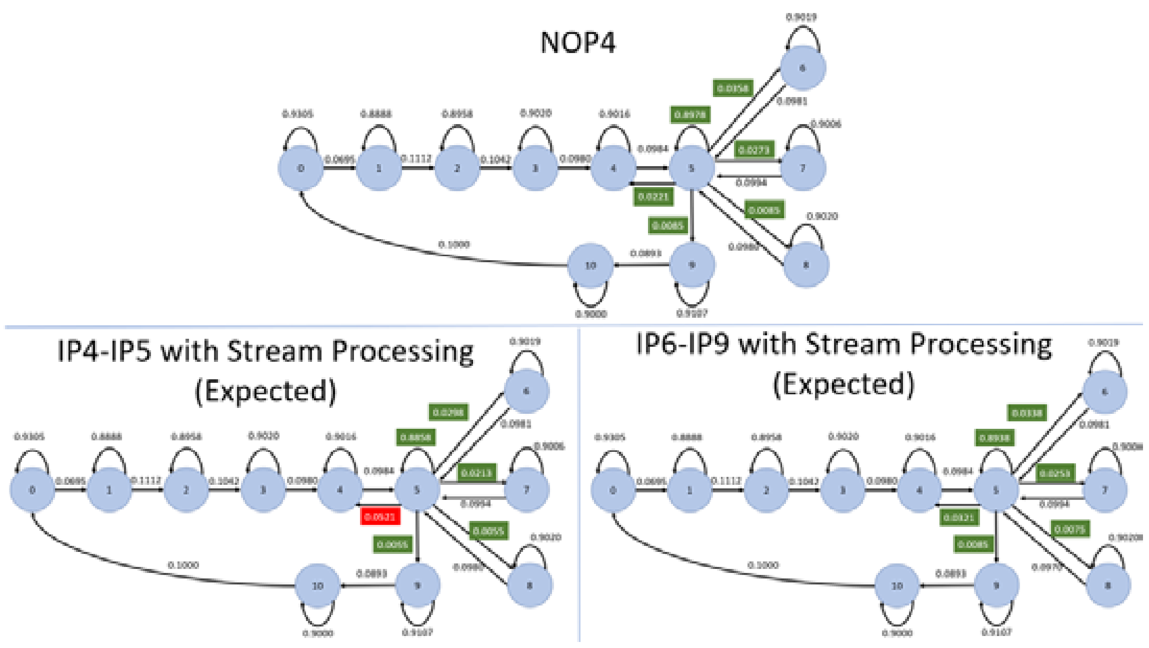
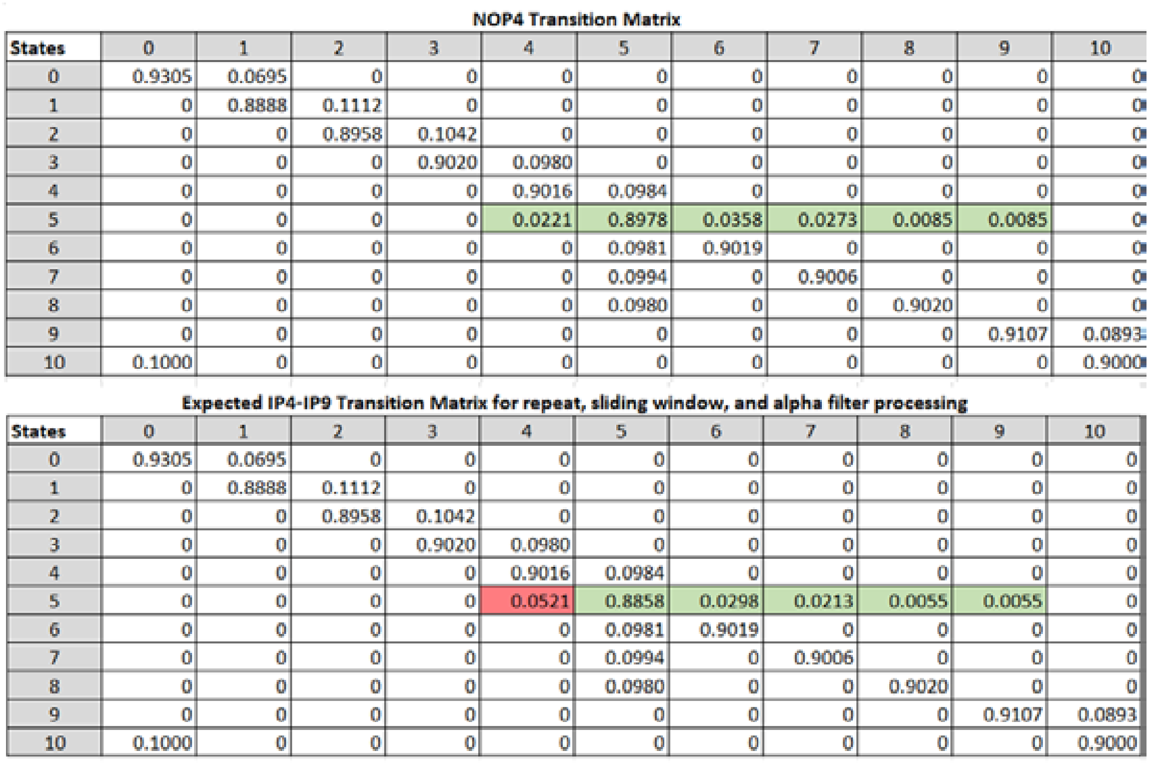

NOP4
IP4-IP5 with Stream Processing (Expected)
IP6-IP9 with Stream Processing (Expected)

Figure 37. Expected state diagrams for IP4-IP9 attacks with stream processing

(3) IP4 through IP9: Repeat, Sliding Window, and Alpha Filter Processing

Repeat processing, sliding window, and alpha filter detection methods limit the number of packets used in each observation model. Therefore, the observation model generated by the packets that contain the localized anomaly is not statistically diluted by iterations of the NOP that contain no anomalies. Hence, each of these detection methods is expected to detect the anomaly in IP4-IP9.

Figures 38 and 39 show the transition matrix and state diagram for the IP4-IP9 attacks when processed with the repeat processing, sliding window, and alpha filter methods. As each processing method is expected to successfully detect the localized anomaly, the expected values of the transition matrix and state diagram for each method have been condensed into a single matrix and diagram. The expected successful detection of the transition anomaly is shown in red, indicating the transition probability that differs with the corresponding NOP4 transition probability by more than Σ. The other transition probabilities leaving state 5 are highlighted in green to show that while they will change, they are not expected to differ from their corresponding probabilities in NOP4 by more than Σ and therefore should not cause anomalies. Additionally, since stream processing involves comparing many different observation models, only the expected observation model that triggers the anomaly is shown. All other generations of the observation model would be redundant since once a single generation detects an anomaly the entire observation capture is considered to contain an anomaly.



NOP4 Transition Matrix

States	0	1	2	3	4	5	6	7	8	9	10
0	0.9305	0.0695	0	0	0	0	0	0	0	0	0
1	0	0.8888	0.1112	0	0	0	0	0	0	0	0
2	0	0	0.8958	0.1042	0	0	0	0	0	0	0
3	0	0	0	0.9020	0.0980	0	0	0	0	0	0
4	0	0	0	0	0.9016	0.0984	0	0	0	0	0
5	0	0	0	0	0.0221	0.8978	0.0358	0.0273	0.0085	0.0085	0
6	0	0	0	0	0	0.0981	0.9019	0	0	0	0
7	0	0	0	0	0	0.0994	0	0.9006	0	0	0
8	0	0	0	0	0	0.0980	0	0	0.9020	0	0
9	0	0	0	0	0	0	0	0	0	0.9107	0.0893
10	0.1000	0	0	0	0	0	0	0	0	0	0.9000

Expected IP4-IP9 Transition Matrix for repeat, sliding window, and alpha filter processing

States	0	1	2	3	4	5	6	7	8	9	10
0	0.9305	0.0695	0	0	0	0	0	0	0	0	0
1	0	0.8888	0.1112	0	0	0	0	0	0	0	0
2	0	0	0.8958	0.1042	0	0	0	0	0	0	0
3	0	0	0	0.9020	0.0980	0	0	0	0	0	0
4	0	0	0	0	0.9016	0.0984	0	0	0	0	0
5	0	0	0	0	0.0521	0.8858	0.0298	0.0213	0.0055	0.0055	0
6	0	0	0	0	0	0.0981	0.9019	0	0	0	0
7	0	0	0	0	0	0.0994	0	0.9006	0	0	0
8	0	0	0	0	0	0.0980	0	0	0.9020	0	0
9	0	0	0	0	0	0	0	0	0	0.9107	0.0893
10	0.1000	0	0	0	0	0	0	0	0	0	0.9000

Figure 38. Expected transition matrixes for IP4 -IP9 attacks with repeat, sliding window, and alpha filter processing methods

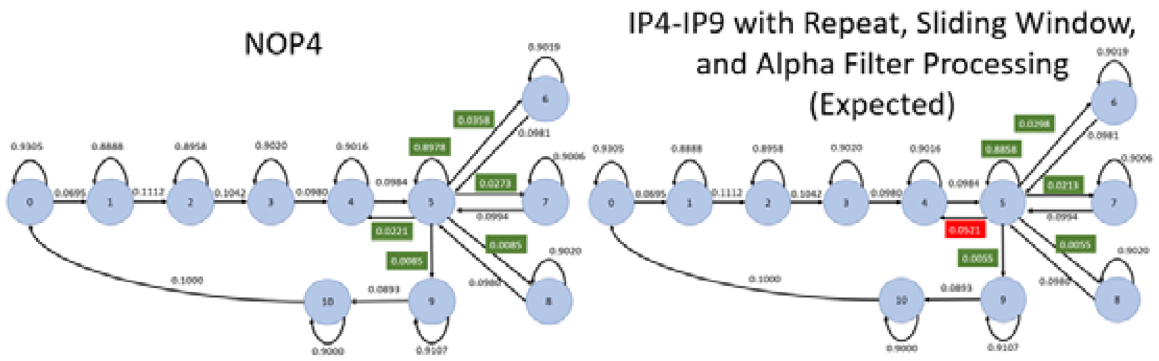

NOP4
IP4-IP9 with Repeat, Sliding Window, and Alpha Filter Processing (Expected)

Figure 39. Expected state diagrams for IP4-IP9 attacks with repeat processing, sliding window, and alpha filter methods

While each of these three methods is expected to correctly detect the probability anomalies, the repeat processing and sliding window methods will likely generate false positives during the final generation of observation model creation and generation, i.e., the final observation model is not guaranteed to be generated from packets that represent a full cycle of the NOP; therefore, it may not contain every state or transition found in the training model. As discussed in Section A.4.d, the alpha filter method is expected to address this performance issue and should reduce the number of false positives. The expected transition matrixes and state diagrams showing the expected false positives for the repeat and sliding window methods are not included, as the exact nature of what states and transitions are affected is difficult to predict.

4. Scalability of data

An operational IDS must process large amounts of data and to measure the performance of the models as the data grow in size, we use two datasets. The small dataset contains relatively small file sizes, e.g., NOP1 contains 5,652 packets and only 599 packets contribute to the generated models. The large dataset contains packet captures made by concatenating existing captures to create test data sets of 10 times and 100 times the size of the original data. Both datasets are used in the same way for observation model generation and training model comparison. For the tests using the large dataset, the expected observation transition matrixes and expected results are the same as for the corresponding small data set tests. Table 13 summarizes the large dataset tests.

Table 13. Large data set captures

Test Name	Generation Method	Corresponding Small Dataset Test	Total Number of Packets	Number of Relevant Packets
IS1x10	Concatenate the IS1 capture with itself 10 times	IS1	466,320	5,510
IS1x100	Concatenate the IS1 capture with itself 100 times	IS1	4,663,200	55,100
IT1x10	Concatenate the IT1 capture with itself 10 times	IT1	389,950	4,620
IT1x100	Concatenate the IT1 capture with itself 100 times	IT1	3,899,500	46,200
IP2x10	Concatenate the IP2 capture with itself 10 times	IP2	678,060	8,020
IP2x100	Concatenate the IP2 capture with itself 100 times	IP2	6,780,600	80,200
IP4x10	Concatenate the IP4 capture with itself 10 times	IP4	772,420	9,150
IP4x100	Concatenate the IP4 capture with itself 100 times	IP4	7,724,200	91,500
IP6x10	Concatenate the IP6 capture with itself 10 times	IP6	778,850	9,210
IP6x100	Concatenate the IP6 capture with itself 100 times	IP6	7,788,500	92,100
IP8x10	Concatenate the IP8 capture with itself 100 times	IP8	771,550	9,130
IP8x100	Concatenate the IP8 capture with itself 100 times	IP8	7,715,500	91,300

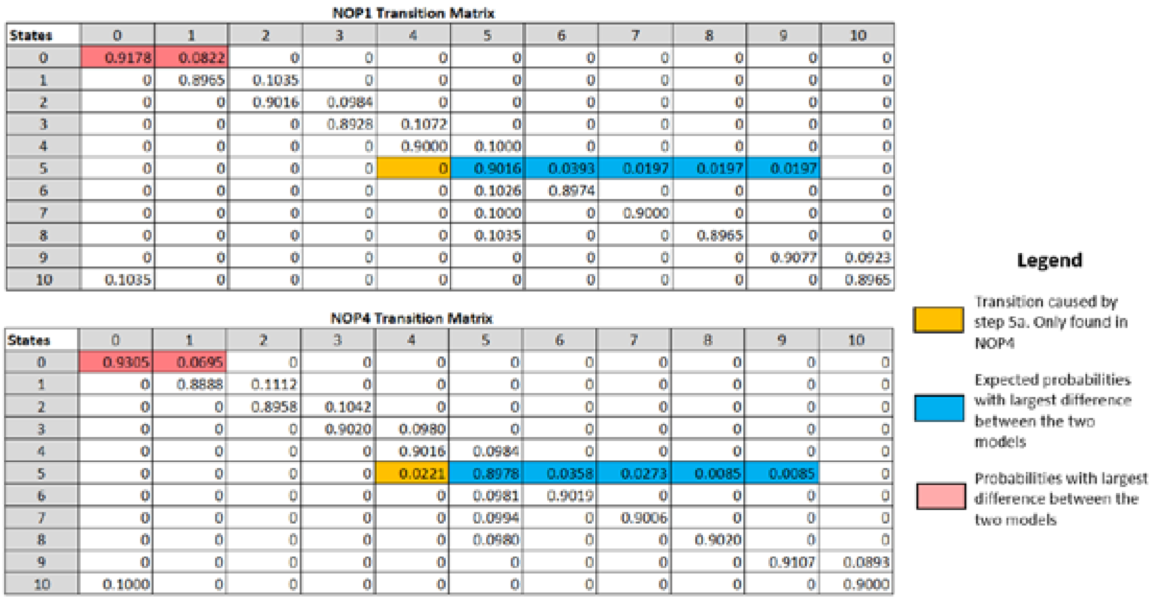
D. INITIAL VALUES FOR HYPER-PARAMETERS

Among the five presented methods for detecting transition probability anomalies, four hyper-parameters are used to control the behavior of these methods. Initial values for these hyper-parameters must be set before performance of the anomaly detection methods can be observed.

1. Probability Tolerance Factor

A probability tolerance factor is used in every probability anomaly detection method. The probability of each transition in the observation model is compared to the probability of that transition in the training model, and if the observation model's probability differs from the training model's by more than Σ, a transition probability error has occurred. Therefore, finding a correct value for Σ is critical to a functioning IDS. The initial value of Σ is set by comparing two of the NOP models together, finding the maximum difference between corresponding transition probabilities, and setting Σ to a value just above that difference. Using the batch processing method with NOP1 as the training model, NOP4 as the observation model, and $\Sigma = 0$ results in a probability anomaly for every transition, but the largest difference in probabilities is 0.0127. Therefore, the initial value for Σ is set at 0.0150. To verify this value, the batch processing method was used with NOP1, NOP2, and NOP3 as the training models, each with NOP4 as the observation model, and no probability anomalies were triggered, indicating that this value does not result in false positives.

Figure 40 shows the transition matrixes of NOP1 (without steps 5a/5b and 8a/8b) and NOP4 (with cycling of steps 5a/5b and 8a/8b), and highlights the probabilities affected by cycling components as well as the greatest difference between corresponding probabilities of the two models. Performing step 5a will cause a transition from state 5 back to state 4, and performing step 8a will cause a transition from state 5 to state 7. Performing step 8b will cause a transition from state 5 to state 6. Since these steps of the cycling both affect transitions leaving state 5, it is expected that the transition with the greatest probability difference between NOP1 and NOP4 will be one of the transition probabilities of leaving state 5, and these are highlighted in blue in Figure 40. Step 5b results in a transition from state 4 to state 5, but since the sequence of transitioning from state 4 to state 4 for five seconds, followed by a single transition from state 4 to state 5 is maintained between both NOP1 and NOP4, the probabilities of the transitions leaving state 4 in NOP 4 are not expected to change in any significant way.



NOP1 Transition Matrix

States	0	1	2	3	4	5	6	7	8	9	10
0	0.9178	0.0822	0	0	0	0	0	0	0	0	0
1	0	0.8965	0.1035	0	0	0	0	0	0	0	0
2	0	0	0.9016	0.0984	0	0	0	0	0	0	0
3	0	0	0	0.8928	0.1072	0	0	0	0	0	0
4	0	0	0	0	0.9000	0.1000	0	0	0	0	0
5	0	0	0	0	0	0.9016	0.0393	0.0197	0.0197	0.0197	0
6	0	0	0	0	0	0.1026	0.8974	0	0	0	0
7	0	0	0	0	0	0.1000	0	0.9000	0	0	0
8	0	0	0	0	0	0.1035	0	0	0.8965	0	0
9	0	0	0	0	0	0	0	0	0	0.9077	0.0923
10	0.1035	0	0	0	0	0	0	0	0	0	0.8965

NOP4 Transition Matrix

States	0	1	2	3	4	5	6	7	8	9	10
0	0.9305	0.0695	0	0	0	0	0	0	0	0	0
1	0	0.8888	0.1112	0	0	0	0	0	0	0	0
2	0	0	0.8958	0.1042	0	0	0	0	0	0	0
3	0	0	0	0.9020	0.0980	0	0	0	0	0	0
4	0	0	0	0	0.9016	0.0984	0	0	0	0	0
5	0	0	0	0	0.0221	0.8978	0.0358	0.0273	0.0085	0.0085	0
6	0	0	0	0	0	0.0981	0.9019	0	0	0	0
7	0	0	0	0	0	0.0994	0	0.9006	0	0	0
8	0	0	0	0	0	0.0980	0	0	0.9020	0	0
9	0	0	0	0	0	0	0	0	0	0.9107	0.0893
10	0.1000	0	0	0	0	0	0	0	0	0	0.9000

Legend

🟨	Transition caused by step 5a. Only found in NOP4
🟦	Expected probabilities with largest difference between the two models
🟥	Probabilities with largest difference between the two models

Figure 40. comparison of NOP1 and NOP4 transition matrixes

Further inspection of the transition matrixes shows an unexpected result. It was expected that the greatest difference between corresponding probabilities of the NOP1 and NOP4 models would be one of the probabilities affected by the component cycling during step 5a/5b or 8a/8b (highlighted in blue in Figure 40). However, the greatest difference of corresponding probabilities is from the transitions out of state 0. Table 14 is an excerpt from Table 7 that shows the shows the AFTL and DIOL component statuses that make up states 0 and 1.

Table 14. DTMC state definitions for states 9 and 10

State Name	Command data hex value	Fill pump status	Drain valve status	Low level alarm	High level alarm	Robot arm command
0	0x580000000000000000000000	OFF	OPEN	LIT	NOT LIT	None
1	0x480000000000000000000000	OFF	SHUT	LIT	NOT LIT	None

The transition between states 0 and 1 is a result of step 1 in the NOP and should not have been significantly affected by the differences between NOP1 and NOP4 captures. While care was taken during the creation of all the traffic captures to spend 5 seconds between each component operation, this deviation is likely due to lack of consistency by the operator when performing step 1 of the NOP. However, this is useful in illustrating the sensitivity of these models to the timing of the steps of the NOP. Since the steps required to create each traffic capture were conducted using a watch to manually time the 5-second delay in between each step, the observed probability difference is considered within tolerance of the system requirements, and therefore the probability tolerance factor Σ remains set at 0.0150.

2. Interval Size

An interval size is used by several probability anomaly detection methods to limit the maximum number of packets used in the creation of an observation model. If the interval size is too large, localized anomalies may not be detected, resulting in false negatives during anomaly detection. If the interval size is too small, then the generated observation model might not be representative of the underlying industrial process and generate false positives during anomaly detection due to missing states or transitions. Therefore, an interval size should at least contain packets representing one full iteration of the industrial process. In order to find this value, the NOP4 capture was observed to see how many packets it took to complete one iteration of the NOP. The greatest number of packets between the start and end of a NOP iteration was 313. Hence, to have interval size be an even number for ease of interaction with the overlap size (discussed in Section D.3), an initial value of 320 for interval size is chosen. The NOP4 capture was used to determine this hyper-parameter because it represents the maximum amount of component cycling allowed, and therefore each iteration requires the most packets to represent.

For the repeat processing, sliding window, and alpha filter methods, two additional tests are conducted with a different interval size. As discussed in section 4.1.6, the repeat processing method of probability anomaly detection is expected to have poor performance if the modeling division made by the interval size bisects the packets that contain the information that creates the probability anomaly in the training model. It is also anticipated that the sliding window and alpha filter methods have consistent performance under these circumstances. Table 15 describes the two scenarios used to test this part of the hypothesis.

Table 15. Interval size performance tests

Test name	**Capture being used**	**Interval size**	**Expected result**
IP6_int1	IP6	206	Repeat method fails to detected probability anomaly. Sliding window and alpha filter successfully detect probability anomaly.
IP8_int1	IP8	175	Repeat method fails to detected probability anomaly. Sliding window and alpha filter successfully detect probability anomaly.

For each of these tests, the interval size is selected to result in a bisection of the portion of packets that contain the anomaly causing data. For IP6_int1, the packets that generate the anomaly are packet numbers 343 through 482; therefore, packet number 412 perfectly bisects the anomaly-causing packets. An interval size 206 will cause the second and third models generated by the repeat processing, sliding window, and alpha filter methods to bisect this portion of anomaly-causing data. For IP8_int1, the packets that generate are packet numbers 597 through 804, therefor packet number 700 perfectly bisects the anomaly-causing packets. An interval size of 175 will cause the fourth and fifth models generated by the repeat processing, sliding window, and alpha filter methods to bisect this portion of the anomaly-causing data. While an interval size of 350 would also result in a bisection of the anomaly- causing data, 175 is preferred because it is closer to the interval size used in IP6_int1; therefore, any effects of smaller than typical interval size will be seen in both of these tests, making their results easier to compare with each other.

3. Window Size

Window size is used by the sliding window detection method to determine how many packets a given observation model should re-use from the previous observation model. Our assumption is that a window size of one half of interval size will allow sufficient packets being included in the overlap between the two observation models to detect any localized anomalies occurring at the interval size division. Additionally, it is small enough to prevent each section of the observation capture from being used in multiple observation models and unnecessarily slowing the performance of the anomaly detection method as a whole. Therefore, an initial value for window size is 160. If the interval size is changed, the window size must also be changed to always be one half of interval size.

4. Alpha Filter Constant

The alpha filter constant represents what percentage of the previous observation model's probability values are used in the current observation model. As discussed in section A.4.d, the alpha filter anomaly detection method is expected to show an increase in performance over the repeat processing and sliding window methods, specifically by not resulting in false positives from the comparison of the final observation model. However, if the alpha filter constant is set too low, the final filtered model will not incorporate enough of the previous filtered model's data to prevent such a false positive. If the filter constant is set to high, the current raw model's probability values will not have enough weight to affect the previous filtered model's probability values when there is an anomaly, resulting in a false negative. An initial value of 0.2 was chosen for the alpha filter constant to give a larger weight to the current raw model's probability values while still providing adjustment from the previous filtered model's probability values. However, a range of alpha filter values are used during tests with the IP6 traffic capture to observe how performance changes as the alpha filter constant changes. Table 16 shows the tests and alpha filter constant values used for this experiment.

Table 16. Alpha filter constant performance tests

Test name	Capture being used	Alpha filter constant	Expected result
IP6_alpha1	IP6	0.05	No performance difference from the repeat and sliding window results from test IP6
IP6_alpha2	IP6	0.1	No performance difference from the repeat and sliding window results from test IP6
IP6_alpha3	IP6	0.2	Successfully detects the probability anomaly, and does not include a false positive from the final raw model
IP6_alpha4	IP6	0.5	Unable to detect any anomalies
IP6_alpha5	IP6	0.7	Unable to detect any anomalies

The IP6 capture is chosen because it contains a localized anomaly that the alpha filter method is designed to be able to detect, and the anomaly is included in the middle of the capture preventing the first filtered model from containing the anomaly. If the first filtered model contains the anomaly, a high alpha filter constant would not result in a false negative because each subsequent generation would consist of a large weighting of the first generation's data.

THIS PAGE INTENTIONALLY LEFT BLANK

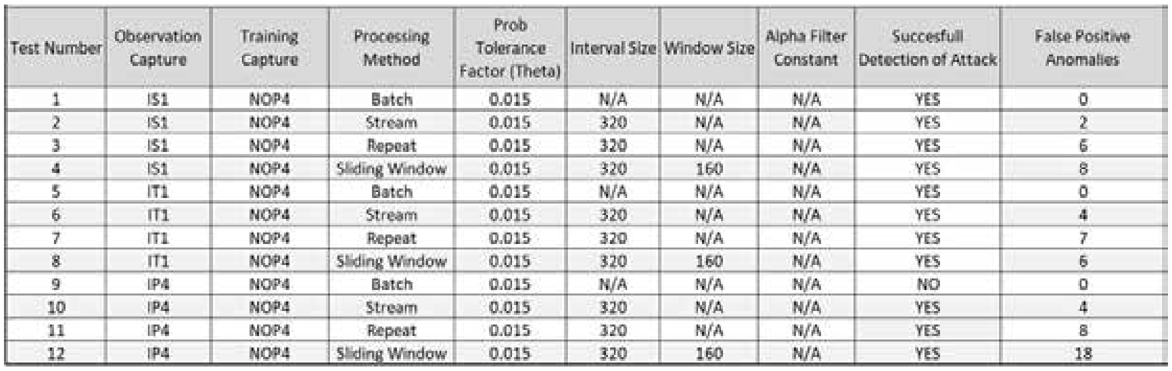
V. RESULTS

This chapter discusses the results for the tests described in Chapter IV. The initial results showed that further tuning of hyper-parameter values was needed. The hyper-parameters of Σ and α were adjusted by optimizing the precision values for select tests, and the optimal values were used for each test described in Chapter IV.

A. INITIAL RESULTS

The tests presented in Chapter IV for the IS1, IS2, and IP4 attacks were conducted first and the results were analyzed to identify potential issues early in the testing process. Table 17 show the summary for these results.

Table 17. Experiment summary for IS1, IS2 and IP4 tests



Test Number	Observation Capture	Training Capture	Processing Method	Prob Tolerance Factor (Theta)	Interval Size	Window Size	Alpha Filter Constant	Succesfull Detection of Attack	False Positive Anomalies
1	IS1	NOP4	Batch	0.015	N/A	N/A	N/A	YES	0
2	IS1	NOP4	Stream	0.015	320	N/A	N/A	YES	2
3	IS1	NOP4	Repeat	0.015	320	N/A	N/A	YES	6
4	IS1	NOP4	Sliding Window	0.015	320	160	N/A	YES	8
5	IT1	NOP4	Batch	0.015	N/A	N/A	N/A	YES	0
6	IT1	NOP4	Stream	0.015	320	N/A	N/A	YES	4
7	IT1	NOP4	Repeat	0.015	320	N/A	N/A	YES	7
8	IT1	NOP4	Sliding Window	0.015	320	160	N/A	YES	6
9	IP4	NOP4	Batch	0.015	N/A	N/A	N/A	NO	0
10	IP4	NOP4	Stream	0.015	320	N/A	N/A	YES	4
11	IP4	NOP4	Repeat	0.015	320	N/A	N/A	YES	8
12	IP4	NOP4	Sliding Window	0.015	320	160	N/A	YES	18

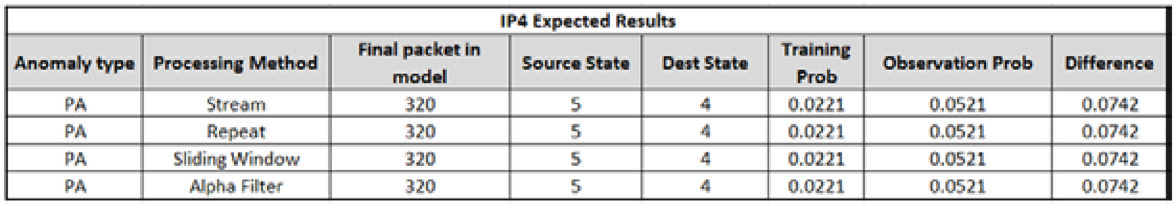
These results matched the expected results relative to detecting the anomaly present in each of the captures. As expected, all four processing methods detected the invalid state anomaly in IS1 (test 1 through 4) and the invalid transition anomaly in IT1 (tests 5 through 8). For test #9, the batch processing method was unable to detect the expected invalid probability anomaly, while tests 10 through 12 show that the stream, repeat, and sliding window methods were able to detect the anomaly.

However, the number of anomalies not related to the attack, particularly from each test using the stream, repeat, and sliding window processing methods, was unexpected, and further investigation of these results was necessary. Of the tests shown in Table 17, tests

9–12 were of particular interest as they contain the highest rate of false positive anomalies. Table 18 shows the expected results for the IP4 tests for each processing methods. The batch processing method was not expected to detect the attack, as the attack only resulted in a localized anomaly. Since the localized anomaly is located early in the capture, the stream, repeat, sliding window, and alpha filter methods were expected to detect the probability anomaly in the first observation model they generated. The attack was expected to be detected by a probability anomaly on the transition from state 5 to state 4. The IP4 attack involves cycling the fill pump more than the NOP specification allows, resulting in the observation model containing more transitions from state 5 to state 4 than training model. Therefore, the transition probability from state 5 to state 4 was expected to be the probability with the greatest change, and the source of the invalid probability anomaly. The expected results for the IP4 tests are compared to the observed results for test 12 shown in Table 19.

In Tables 18 and 19, the value in the "final packet in model" column denotes the number of the final packet used in the creation of the observation model that observed the anomaly. This enables tracking of which portions of the observation capture that generated the anomaly when detecting localized anomalies.

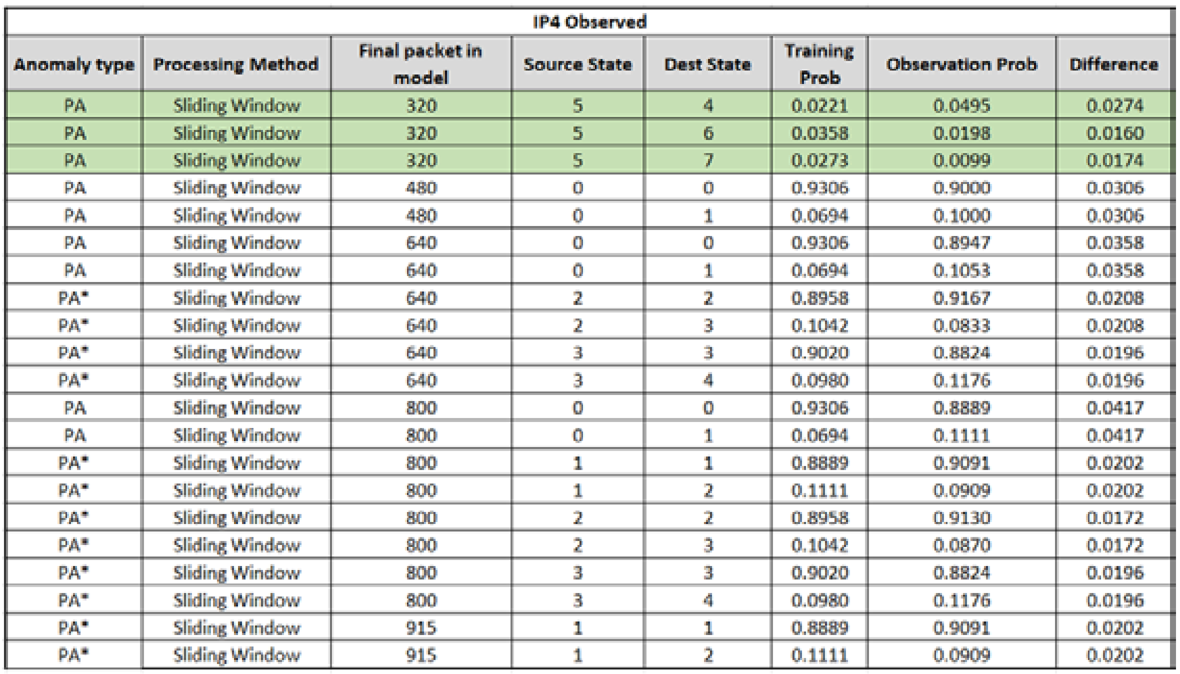
Table 18. Expected results for IP4



IP4 Expected Results							
Anomaly type	Processing Method	Final packet in model	Source State	Dest State	Training Prob	Observation Prob	Difference
PA	Stream	320	5	4	0.0221	0.0521	0.0742
PA	Repeat	320	5	4	0.0221	0.0521	0.0742
PA	Sliding Window	320	5	4	0.0221	0.0521	0.0742
PA	Alpha Filter	320	5	4	0.0221	0.0521	0.0742

PA = Probability Anomaly

Table 19. Observed results for test 12 (IP4 with sliding window processing)



IP4 Observed							
Anomaly type	Processing Method	Final packet in model	Source State	Dest State	Training Prob	Observation Prob	Difference
PA	Sliding Window	320	5	4	0.0221	0.0495	0.0274
PA	Sliding Window	320	5	6	0.0358	0.0198	0.0160
PA	Sliding Window	320	5	7	0.0273	0.0099	0.0174
PA	Sliding Window	480	0	0	0.9306	0.9000	0.0306
PA	Sliding Window	480	0	1	0.0694	0.1000	0.0306
PA	Sliding Window	640	0	0	0.9306	0.8947	0.0358
PA	Sliding Window	640	0	1	0.0694	0.1053	0.0358
PA*	Sliding Window	640	2	2	0.8958	0.9167	0.0208
PA*	Sliding Window	640	2	3	0.1042	0.0833	0.0208
PA*	Sliding Window	640	3	3	0.9020	0.8824	0.0196
PA*	Sliding Window	640	3	4	0.0980	0.1176	0.0196
PA	Sliding Window	800	0	0	0.9306	0.8889	0.0417
PA	Sliding Window	800	0	1	0.0694	0.1111	0.0417
PA*	Sliding Window	800	1	1	0.8889	0.9091	0.0202
PA*	Sliding Window	800	1	2	0.1111	0.0909	0.0202
PA*	Sliding Window	800	2	2	0.8958	0.9130	0.0172
PA*	Sliding Window	800	2	3	0.1042	0.0870	0.0172
PA*	Sliding Window	800	3	3	0.9020	0.8824	0.0196
PA*	Sliding Window	800	3	4	0.0980	0.1176	0.0196
PA*	Sliding Window	915	1	1	0.8889	0.9091	0.0202
PA*	Sliding Window	915	1	2	0.1111	0.0909	0.0202

PA = Probability Anomaly

PA* = Probability Anomaly that would not exists if Σ=0.024

A comparison of the expected and observed results shows two things. First, the anomaly was successfully detected. While the observed results also show anomalous transitions from state 5 to states 6 and 7, this is not alarming. Since the sum of all probabilities leaving a state must be 1.0 [15], changing the probability of the state 5 to state 4 transition is guaranteed to change all the other probabilities for transitions leaving state 5. In this case, the probabilities of transitions from state 5 to states 6 and 7 were changed enough to be considered as anomalies. These are not considered false positives because they are a direct result of the true anomaly and would be considered a successful detection of the attack, and are highlighted in green in Table 19 to reflect that they are part of the successful attack detection.

However, the second feature shown by analyzing the results is the high number of false positive anomalies. Eighteen of the detected anomalies are not the result of the attack, and while some number of false positive anomalies were expected for the final model generated by the repeat and sliding window methods, these false positive anomalies are from observation models made from all points within the observation capture. Additionally, this is not isolated to just test 12, and false positive probability anomalies are a common feature for every test using the streaming, repeat, and sliding window processing methods.

Therefore, the conclusion drawn from these preliminary tests is that while anomaly detection is performing as expected, the number of false positive anomalies must be addressed before a comprehensive review of the results is conducted. The expected cause of the observed high number of false positive anomalies is the value of the probability tolerance factor (Σ). Analysis of Table 19 shows that the false positive anomalies marked with an asterisk could be eliminated by increasing Σ to 0.027 while retaining the ability to detect the attack.

B. HYPER-PARAMETER ADJUSTMENT

This section describes the performance metrics and tests used to adjust the hyper-parameter values of Σ, as well as the results of the IP6_alpha series of tests that determine an optimal value for the alpha filter constant.

1. Probability Tolerance Factor (Σ) Tuning

As described in Chapter III, Σ was set by comparing the batch models generated using the NOP series of captures. By definition, the NOP series of captures do not contain anomalies; therefore, Σ was set at a value just above the maximum difference between corresponding probabilities in these models. However, after running the initial tests, the effect of changing Σ can be measured on captures containing anomalies. The design goals for an optimal value of Σ are to 1) maximize detection of the attack and 2) minimize the number of anomalies that are not a result of the attack.

a. Methodology

In order to meet these goals, the repeat and sliding window detection algorithms were run with Σ values ranging from 0.0 to 0.06. The number of True Positive (TP) and False Positive (FP) results for each run were observed, and the precision value for each calculated. A value for Σ was then chosen that maximized precision for all the detection algorithms. The equation for calculating precision [18] is shown in Eq. 2:

$$Precision = \frac{True\ Positives}{(True\ Positives + False\ Positives)} \qquad (2)[18]$$

To use precision, a method for counting TP and FP must be defined. The goal of the algorithms is to observe at least one probability anomaly due to the effects of the attacks present in each of the captures, and the attack is considered detected if one or more generated anomalies are the result of the attack. Therefore, when analyzing the anomalies observed by a detection method, if one or more of the anomalies are the result of the attack, true positives is set to 1. If none of the generated anomalies are a result of the attack, true positives are set to 0. False positives are determined by the number of generated anomalies that are not a result of the attack.

It is worth noting that typically the tradeoffs between true positives, false positives, true negatives, and false negatives are show by comparing precision with recall [18]. However, since there is only one attack to detect in each capture, the maximum value for true positive is 1. Max value for false negative is also 1. Additionally, the attack is exclusive, i.e., if true positives are 1, false negatives must be 0, and if true positives are 0, false positives must be 1. When these limitations are observed with the recall equation shown in equation 3, it becomes apparent that the recall statistic does not add any new information in this case.

$$Recall = \frac{True\ Positives}{(True\ Positives + False\ Negatives)} \qquad (3)[18]$$

Since if true positives are 1, false negatives must be 0, recall is always 1 if the anomaly is detected, and always 0 if the anomaly is not detected. Precision is always some positive value if the anomaly is detected (true positive is 1), or either 0 or undefined if the anomaly is not detected (true positive is 0). Therefore, maximizing precision is sufficient for accomplishing both design goals and considering recall would not cause any change in the optimal value for Σ.

b. Test Selection and Results

All of the false positive anomalies observed in the initial testing were are probability anomalies; therefore, only the IP series of tests were used for tuning the value of Σ, The methodology for optimizing Σ was performed using the IP2 and IP4 tests for the repeat and sliding window processing methods described in CH IV. An optimal value for Σ was chosen by comparing the results of these four tests as shown in Figure 41. Choosing IP2 and IP6 for Σ tuning provides one traffic capture with the attack present throughout the entire capture (IP2), and one where the attack results in a localized anomaly (IP6). Each capture was used with both the repeat and sliding window processing methods, as these are the two processing methods that generated the highest rate of false positive anomalies. Each of these four tests were run with a range of values for Σ, starting at Σ = 0 and incrementing by 0.002 until Σ = 0.06. The resulting precision values are shown in Figure 41. It is also worth noting that as the value for Σ increases, both the number of true positives and false positives could be 0.

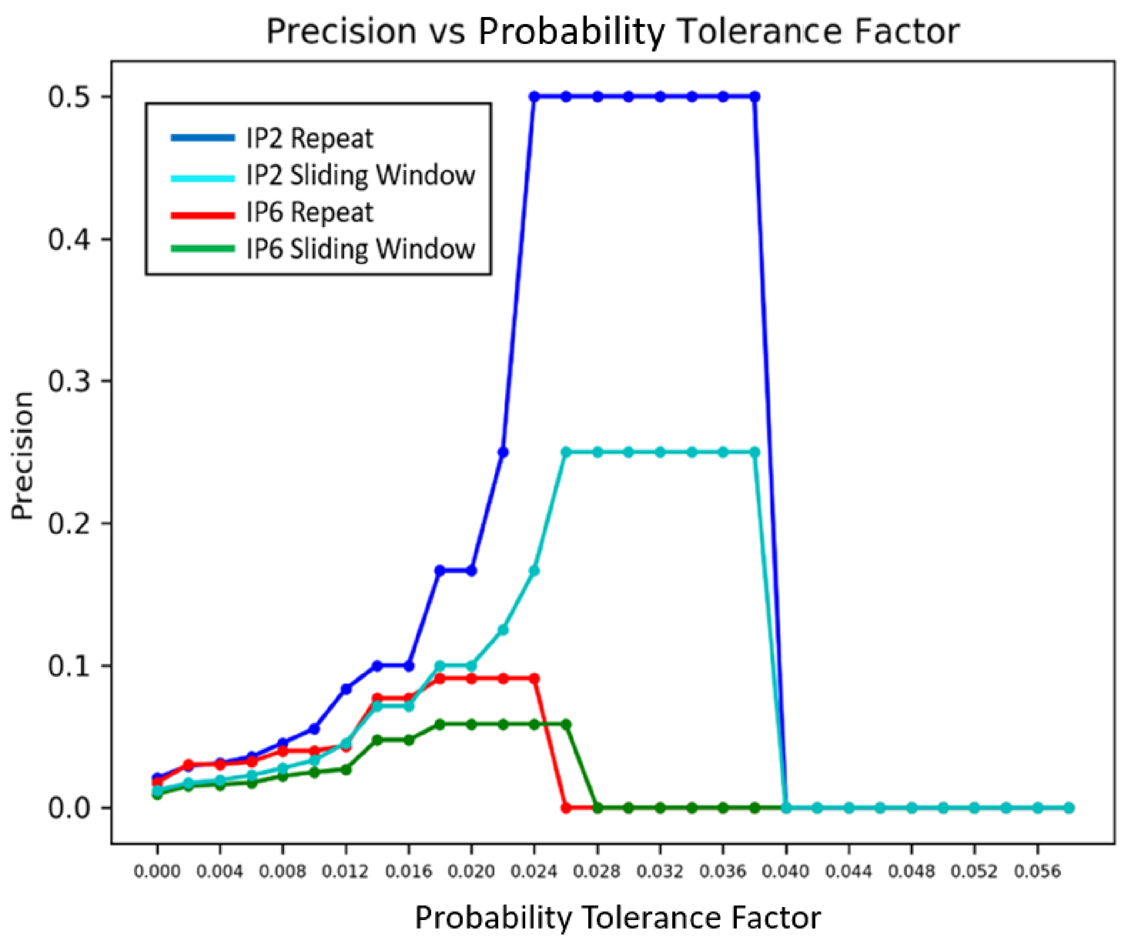


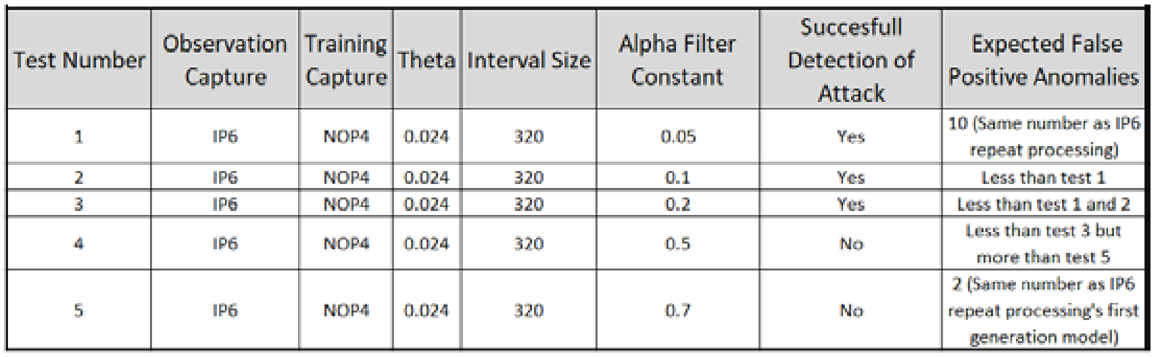
Figure 41. Precision vs. probability tolerance factor results

A Σ value of 0.024 produced the maximum precision value for the IP2 repeat, IP6 repeat, and IP6 sliding window tests, and the second highest value for the IP2 sliding window test. While increasing Σ to 0.026 resulted in the maximum precision values for the IP2 repeat, IP2 sliding window, and IP6 sliding window tests, it also caused the IP6 repeat test to fail detecting the attack (shown by a precision value of 0). Since the highest priority design goal is to eliminate false negatives, a Σ value of 0.024 was chosen as the optimal value and used in all further tests.

2. Alpha Filter Constant Tuning

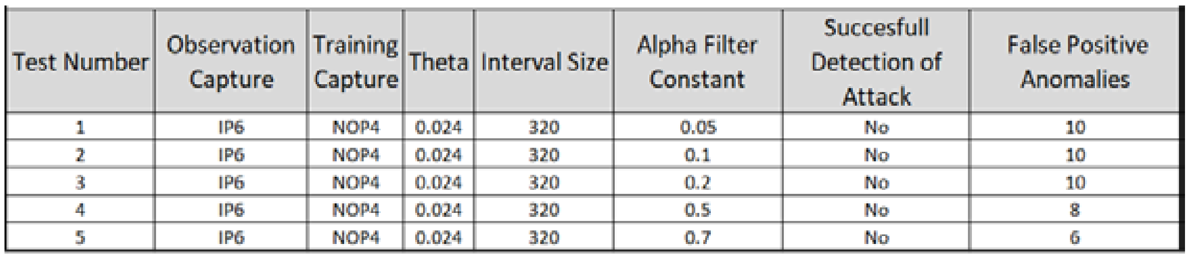
As described in Chapter IV, the alpha filter constant α is tuned by observing performance for a range of values and selecting an alpha filter constant value that results in consistent detection of the attack while minimizing false positive anomalies. The observation traffic capture is split into segments by the interval size and then multiple generations of observation models are created using the DTMC modeling algorithm. In the repeat processing method, each of these generations of models are completely independent from each other, but the alpha filter processing method creates a filtered model by taking a weighted average of the previous generation's filtered model's transition matrix and the current generation's unfiltered model's transition matrix. The alpha filter constant determines the weight that is placed upon the previous generation's transition matrix when computing this weighted average. As the alpha filter constant value approaches 0, each previous generation's model has less effect than the current generation's filtered model, until each generation is fully independent of the last generation and performance matches that of the repeat processing method. As the alpha filter constant value approaches 1, the previous generation's model begins to dominate the current generation's filtered model until each generation's filtered model is a copy of the first generation's model. Therefore, attacks are only detected if they are present in the first observation model created by the detection algorithm. The IP6 capture was chosen for this analysis because it contains a localized anomaly that the alpha filter method is expected to detect, and the anomaly is included in the middle of the capture to prevent the first filtered model from observing the anomaly. Table 20 shows the range of alpha filter constant values that were tested and the expected results, and Table 21 shows a summary of the actual results.

Table 20. Expected results of alpha filter constant testing



Test Number	Observation Capture	Training Capture	Theta	Interval Size	Alpha Filter Constant	Succesfull Detection of Attack	Expected False Positive Anomalies
1	IP6	NOP4	0.024	320	0.05	Yes	10 (Same number as IP6 repeat processing)
2	IP6	NOP4	0.024	320	0.1	Yes	Less than test 1
3	IP6	NOP4	0.024	320	0.2	Yes	Less than test 1 and 2
4	IP6	NOP4	0.024	320	0.5	No	Less than test 3 but more than test 5
5	IP6	NOP4	0.024	320	0.7	No	2 (Same number as IP6 repeat processing's first generation model)

Table 21. Results of alpha filter constant testing



Test Number	Observation Capture	Training Capture	Theta	Interval Size	Alpha Filter Constant	Succesfull Detection of Attack	False Positive Anomalies
1	IP6	NOP4	0.024	320	0.05	No	10
2	IP6	NOP4	0.024	320	0.1	No	10
3	IP6	NOP4	0.024	320	0.2	No	10
4	IP6	NOP4	0.024	320	0.5	No	8
5	IP6	NOP4	0.024	320	0.7	No	6

Unexpectedly, even an alpha filter constant of 0.05 resulted in failure to detect the attack. Further inspection of each generation of transition matrixes for test number 1 gave details as to why. Figure 42 shows the transition matrix for each generation as well as the transition matrix for NOP4 for comparison.

NOP4 Transition Matrix

States	0	1	2	3	4	5	6	7	8	9	10
0	0.9305	0.0695	0	0	0	0	0	0	0	0	0
1	0	0.8888	0.1112	0	0	0	0	0	0	0	0
2	0	0	0.8958	0.1042	0	0	0	0	0	0	0
3	0	0	0	0.9020	0.0980	0	0	0	0	0	0
4	0	0	0	0	0.9016	0.0984	0	0	0	0	0
5	0	0	0	0	0.0221	0.8978	0.0358	0.0273	0.0085	0.0085	0
6	0	0	0	0	0	0.0981	0.9019	0	0	0	0
7	0	0	0	0	0	0.0994	0	0.9006	0	0	0
8	0	0	0	0	0	0.0980	0	0	0.9020	0	0
9	0	0	0	0	0	0	0	0	0	0.9107	0.0893
10	0.1000	0	0	0	0	0	0	0	0	0	0.9000

IP6 Transition Matrix for Alpha Filter, packets 0 - 320, alpha constant = 0.05

States	0	1	2	3	4	5	6	7	8	9	10
0	0.91304	0.08695	0	0	0	0	0	0	0	0	0
1	0	0.88888	0.11111	0	0	0	0	0	0	0	0
2	0	0	0.90909	0.0909	0	0	0	0	0	0	0
3	0	0	0	0.9	0.1	0	0	0	0	0	0
4	0	0	0	0	0.89473	0.10526	0	0	0	0	0
5	0	0	0	0	0	0.89473	0.0421	0.02105	0.02105	0.02105	0
6	0	0	0	0	0	0.09756	0.90243	0	0	0	0
7	0	0	0	0	0	0.09523	0	0.90476	0	0	0
8	0	0	0	0	0	0.0909	0	0	0.90909	0	0
9	0	0	0	0	0	0	0	0	0	0.90909	0.0909
10	0.0625	0	0	0	0	0	0	0	0	0	0.9375

IP6 Transition Matrix for Alpha Filter, packets 321 - 640, alpha constant = 0.05

States	0	1	2	3	4	5	6	7	8	9	10
0	0.89009	0.1099	0	0	0	0	0	0	0	0	0
1	0	0.89944	0.10055	0	0	0	0	0	0	0	0
2	0	0	0.90909	0.0909	0	0	0	0	0	0	0
3	0	0	0	0.88944	0.11055	0	0	0	0	0	0
4	0	0	0	0	0.90237	0.09762	0	0	0	0	0
5	0	0	0	0	0.04523	0.90426	0.0202	0.0101	0.0101	0.0101	0
6	0	0	0	0	0	0.09987	0.90012	0	0	0	0
7	0	0	0	0	0	0.09976	0	0.90023	0	0	0
8	0	0	0	0	0	0.09954	0	0	0.90045	0	0
9	0	0	0	0	0	0	0	0	0	0.90909	0.0909
10	0.14927	0	0	0	0	0	0	0	0	0	0.85072

IP6 Transition Matrix for Alpha Filter, packets 641 - 921, alpha constant = 0.05

States	0	1	2	3	4	5	6	7	8	9	10
0	0.8995	0.10049	0	0	0	0	0	0	0	0	0
1	0	0.93559	0.0644	0	0	0	0	0	0	0	0
2	0	0	0.90909	0.0909	0	0	0	0	0	0	0
3	0	0	0	0.89947	0.10052	0	0	0	0	0	0
4	0	0	0	0	0.88956	0.11043	0	0	0	0	0
5	0	0	0	0	0.0023	0.88847	0.0437	0.02185	0.02185	0.02185	0
6	0	0	0	0	0	0.09767	0.90232	0	0	0	0
7	0	0	0	0	0	0.09998	0	0.90001	0	0	0
8	0	0	0	0	0	0.09134	0	0	0.90865	0	0
9	0	0	0	0	0	0	0	0	0	0.90909	0.0909
10	0.10246	0	0	0	0	0	0	0	0	0	0.89753

Figure 42. Alpha filter test transition matrixes

The red highlighted field in each of the matrixes shown in Figure 42 shows the transition probability for the transition from state 5 to state 4, and this transition probability is expected to cause the probability anomaly. This is the transition probability that most affected by the attack present in packets 321–640, and needs to differ from the corresponding transition probability in NOP4 by more than Σ = 0.024 to be detected. However, even with alpha constant value of 0.05, the difference between the filtered model's transition probability and NOP4's transition probability is 0.023; therefore, no anomaly was detected.

The alpha filter processing method relies on two hyper-parameters, and both must be tuned simultaneously to achieve optimal results. The value of α affects the probability values in the filtered model, and the value of Σ determines if that probability is considered an anomaly. Therefore, the values for Σ and α must be tuned jointly to obtain the most effective result. For the same reasons discussed in Section V.B.1, precision was used as the classifier to measure performance. This test used values for α ranging from 0.1 to 0.9, increasing by 0.1, and values for Σ ranging from 0 to 0.6, increasing by 0.002. As with the Σ tuning in section V.B.1, this test was run on both IP2 and IP6 captures to provide data with attacks that are consistent throughout the capture and attacks that result in a localized anomaly. The contour plots for the IP6 and IP2 tests (Figure 43) show the precision values for each α and Σ pair as well as highlight the maximum precision value.

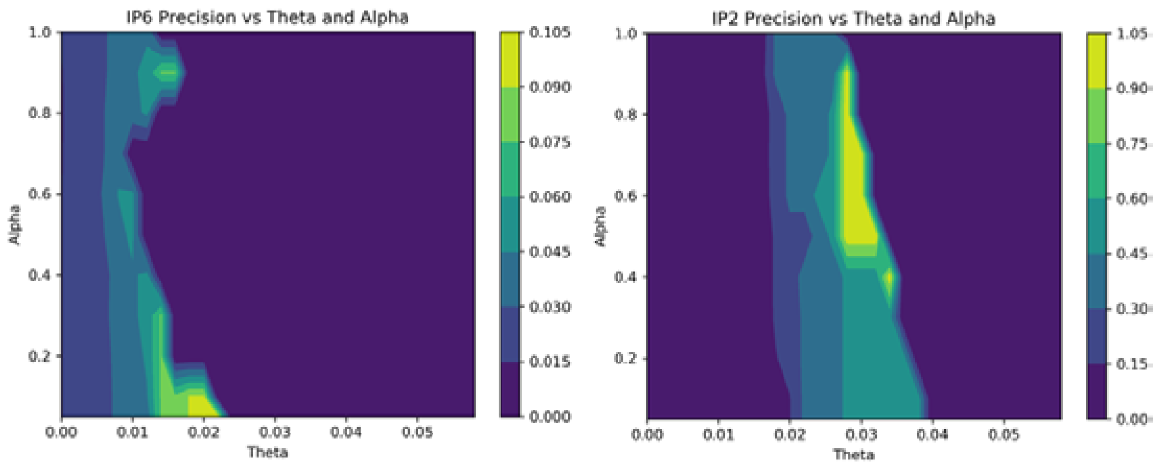


Figure 43. Contour plots of precision vs. theta and alpha for IP6 and IP2

The maximum precision value for the IP6 test was 0.091 and occurred when α = 0.1 and Σ = 0.018 – 0.022, and is highlighted in bright yellow on the contour plot. The maximum precision value for the IP2 test was 1.0 and occurred over multiple points, including when α = 0.4 and Σ = 0.034, α = 0.5 and Σ = 0.028-0.032, and α = 0.6 and Σ = 0.028 – 0.030. This shows a large divergence in the optimal hyper-parameter settings between attacks that persist throughout the observation capture and attacks that result in a localized anomaly. In fact, the optimized α and Σ values from the IP2 resulted in failure to detect the attack present in the IP6 test.

While it was expected that the alpha filter method using optimized hyper-parameters would be able to detect attacks that result in a localized anomaly, the results showing worse alpha filter processing performance for attacks that resulted in a localized anomaly were not unexpected. The objective of applying the alpha filter is to reduce the variance in transition probabilities between each generation of model created by the detection algorithm. It was hoped that this variance reduction would be sufficient to reduce the number of false positives created by slight deviations between operations of the industrial cycle, while still retaining the capability to detect attacks that result in localized anomalies. However, the IP6 (localized anomaly in middle of capture) optimization test showed this was not the case as the optimal α value was a close to 0, resulting in performance that approached the operation and results of the repeat processing method. Additionally, the optimal Σ value shown in the IP6 α-and-Σ tuning test (0.022) was close to the optimized Σ value shown in section V.B.1 (0.024) and the maximum precision from this test (0.091) was the same as the maximum precision shown for the repeat processing method in section V.B.1 (0.091). Therefore, the alpha filter processing method did not provide any performance improvement over the repeat processing method when detecting attacks that resulted in localized anomalies.

The alpha filter method did show performance improvement in other areas. The IP2 precision vs. Σ-and-α test shown in Figure 43 shows a maximum precision value of 1.0, indicating successful detection of the attack with no false positives. The optimal repeat processing performance shown in section V.B.1 has a precision value of 0.5, indicating successful detection of the attack along with the presence of a false positive anomaly. This suggests that when detecting an attack that is present throughout the capture the alpha filter method does offer the expected performance improvements. Therefore, when using the alpha filter method in further testing, values of α = 0.5 and Σ = 0.03 were used. These values were selected as they represent the median of the optimal range of values of α = 0.4 – 0.6 and Σ = 0.28 – 0.32.

C. UPDATED RESULTS

Once the hyper-parameters for both the single-variable tests (batch, stream, repeat, sliding window) and the dual-variable test (alpha filter) have been optimized, the full set of tests presented in Chapter IV were performed. When observing results for each of these tests, two main factors were considered. First, if the attack was successfully detected, the result was considered a true positive. Second, each probability anomaly generated by a processing method would increase the count of the false positives for that test. As discussed in section V.B.1, precision is a metric that measures both of these, and since there in only one attack in each capture, the traditional method monitoring recall does not add any information to the comparative analysis. Performance for each of five anomaly detection methods were compared by observing the precision scores for each test.

1. IS1 and IS2 Attacks

Both the IS1 and IS2 attack are expected to create one or more new states in the observation model, and each detection algorithm is expected to detect these invalid states. Additionally, the creation of a new state in the observation model must result in the creation of at least one new transition and will possibly affect the probabilities of transitions to valid states. Therefore, transition and probability anomalies are expected to be generated as a result of the attack as well. Figure 44 shows the performance of each of the processing methods for the IS1 and IS2 attacks.

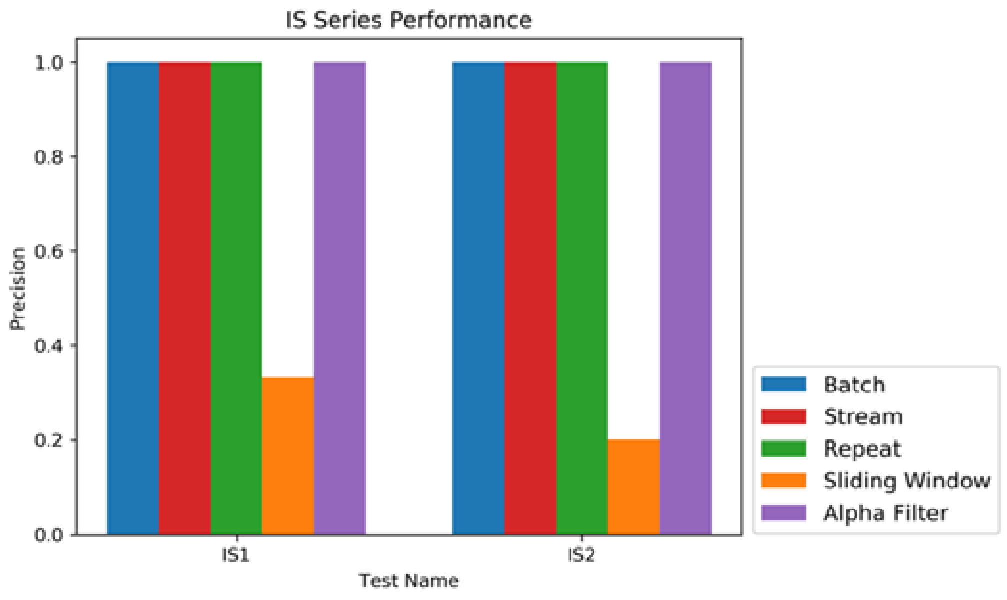


Figure 44. IS1 and IS2 attack detection results

As shown in Figure 44, for the batch, stream, repeat, and alpha filter processing methods, each detected the attacks in both IS captures and have a precision value of 1.0, indicated no false positive anomalies. The sliding window processing method did successfully detect both attacks but did also have unexpected false positive anomalies. For the IS1 attack, the sliding window method returned two false positive anomalies, and for the IS2 attack it returned four false positive anomalies. Each of these false positive anomalies is a probability anomaly. With the exception of false positive anomalies from the sliding window processing method, each anomaly detection method performed as expected for each of these attacks.

2. IT1 and IT2 Attacks

Both IT1 and IT2 attacks are expected to result in creating a new transition in the observation models, and each detection algorithm is expected to successfully detect these attacks. Figure 45 shows the performance of the processing methods for the IS1 and IS2 attacks.

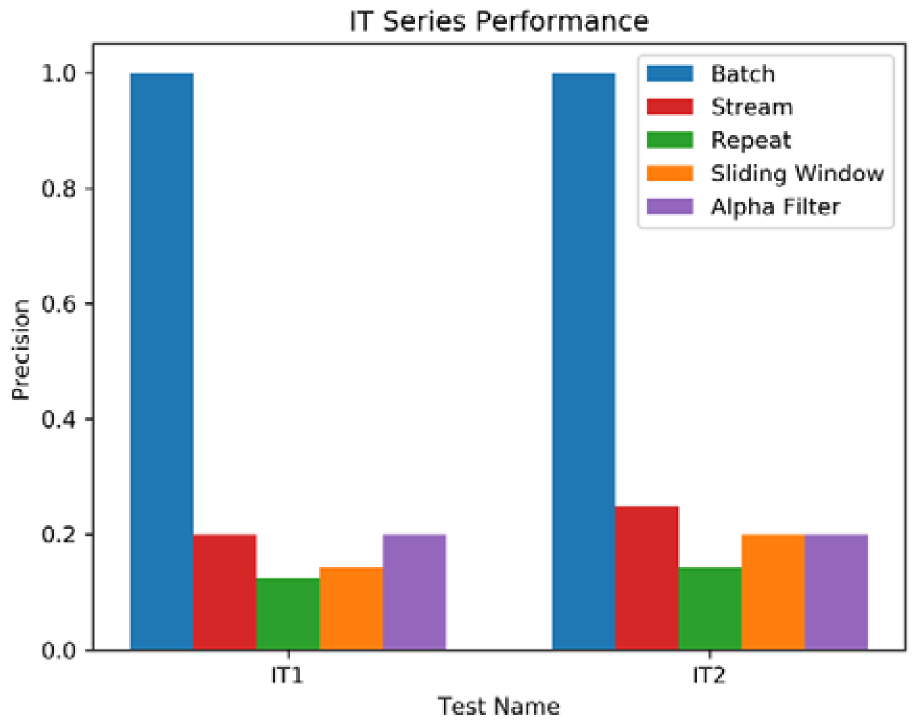


Figure 45. IT1 and IT2 attack detection results

As shown in Figure 45, each processing method did detect the attack, however only the batch processing method did so without any false positive anomalies for either attack. The stream processing method was the second-best performing algorithm, resulting in four false positives for the IT1 attack, and three false positives for the IT2 attack. The alpha filter processing method had similar performance to the stream method with four false positives generated for each attack. The repeat method had the worst performance in both attacks, with seven false positives anomalies in the IT1 attack and six in the IT2 attack.

3. IP1, IP2 and IP3 Attacks

The IP1-IP3 attacks are designed to cause a probability anomaly in their observation model by cycling the fill pump more than the NOP specification allows. All of them perform this cycling in each iteration of their respective procedure, but each test consists of a differing number of cycles in the attack. IP1 cycles the fill pump three times, IP2 cycles the fill pump five time, and IP3 cycles the fill pump ten time. Therefore, the IP1 attack is expected to be the most difficult to detect, and the IP3 attack should be the easiest to detect. Figure 46 shows the performance of the processing methods for the IP1, IP2 and IP3 attacks.

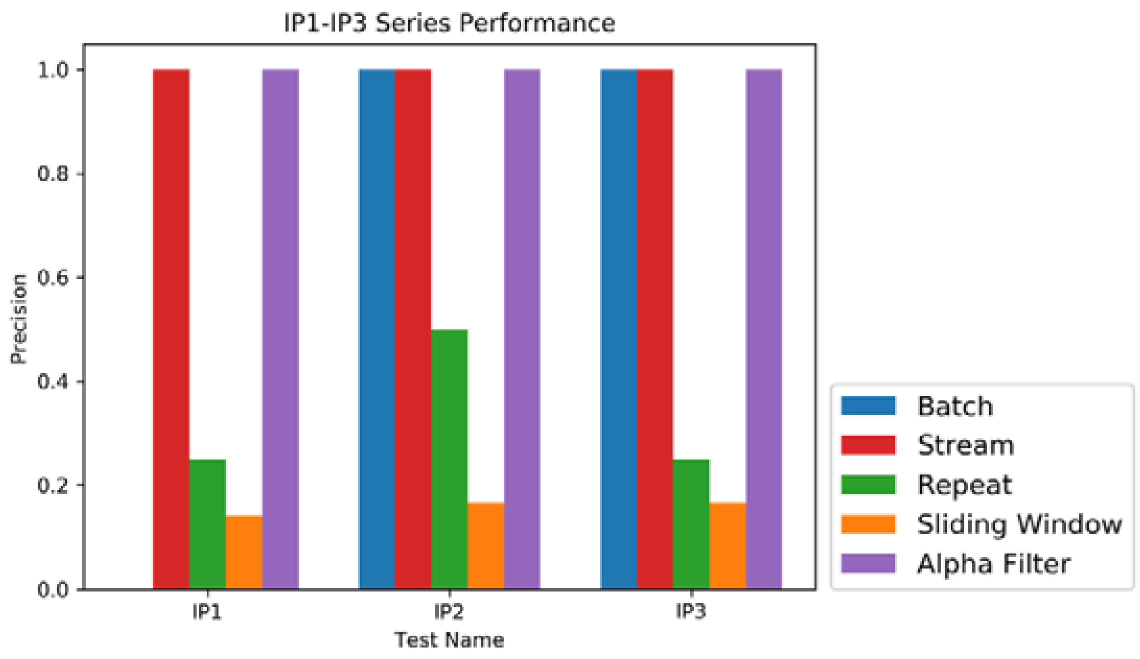


Figure 46. IP1, IP2, and IP3 attack detection results

IP1 is the first attack where a processing method failed to detect the attack. While the batch processing method did successfully detect the IP2 and IP3 attacks with no false positive anomalies, it failed to detect the IP1 attack. However, both the stream and alpha filter methods detected each attack with no false positive anomalies. Additionally, the repeat and sliding window algorithms successfully detected each attack, but also included several unexpected false positive anomalies.

These results begin to confirm one of the hypotheses about the detection methods. While the batch processing method is able to successfully detect obvious attacks (invalid state, invalid transition) with minimal false positive anomalies, it can fail to detect more difficult types of attacks, as shown in IP1.

4. IP4-IP9 Attacks

The IP4 through IP9 attacks are intended to create a probability anomaly by cycling the fill pump more than the NOP specification allows. Additionally, each of the attacks only occur in one of the five iterations of the NOP that make up the observation capture. This is intended to test each processing method's ability to detect localized anomalies. IP4 and IP5 conduct the attack in the first of the five iterations of the NOP, IP6 and IP7 conduct the attack in the third of the five iterations of the NOP, and IP8 and IP9 conduct the attack in the fifth of the five iterations in the NOP. The attack in IP4, IP6, and IP8 consists of five cycles of the fill pump, while the attack in IP5, IP7, and IP9 consists of ten cycles of the fill pump. Figure 47 shows the performance of the processing methods for the IP4-IP9 attacks.

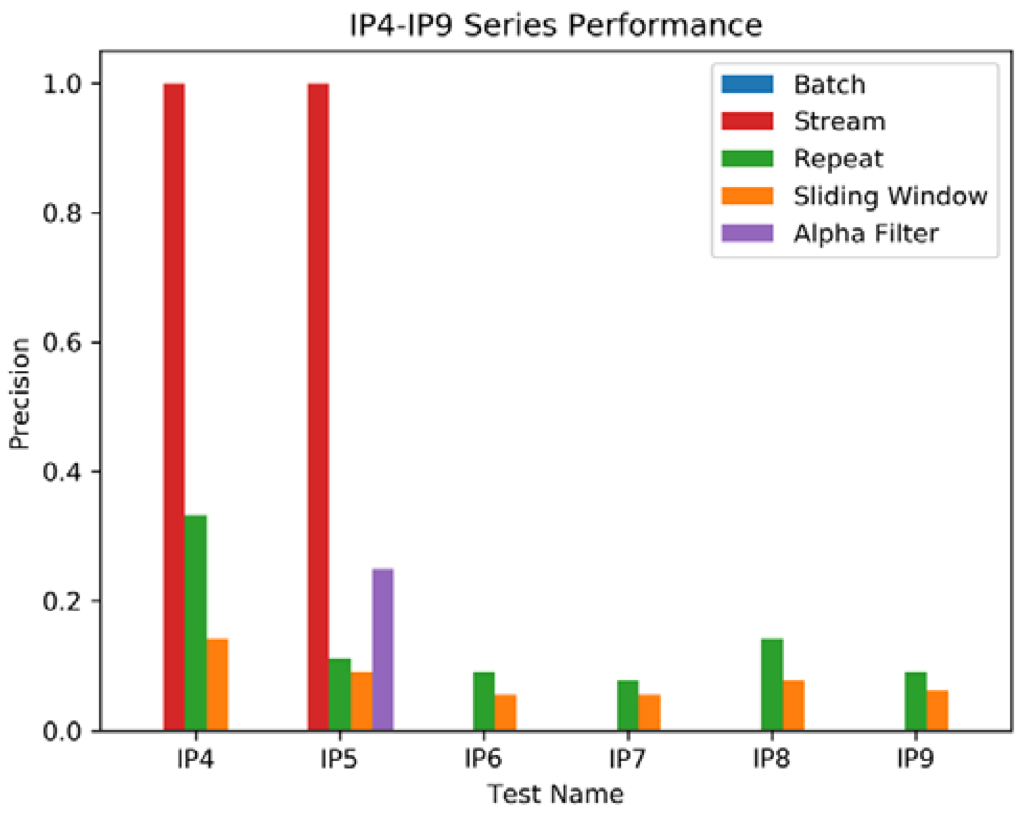


Figure 47. IP4-IP9 attack detection results

As expected, the batch processing method failed to detect any of these attacks, and the stream processing method was only able to detect the attacks that resulted in a localized anomaly at the beginning of the capture (IP4 and IP5). Both the repeat and sliding window methods were able to detect every attack but produced a considerable number of false positive anomalies. The alpha filter method was able to detect the IP5 attack but failed to detect any of the other attacks. Based on the decisions made when tuning the alpha filter hyper-parameters, the inconsistent performance when detecting attacks that result in localized anomalies is expected.

5. Interval Size Testing

The IP6_int1 and IP8_int1 attacks are designed to compare the performance of the repeat and sliding window methods when the interval size causes the division between two of the observation models created by the processing algorithm to bisect the packets that contain the attack in the observation capture. This is expected to cause the attack on the observation models to be statistically diluted, resulting in the repeat processing method failing to detect the attack. The sliding window method creates more overlap between what portions of the observation capture are used to generate the observation model and is expected to detect these attacks. The results of the IP6_int1 and IP8_int1 attacks are shown in Figure 48.

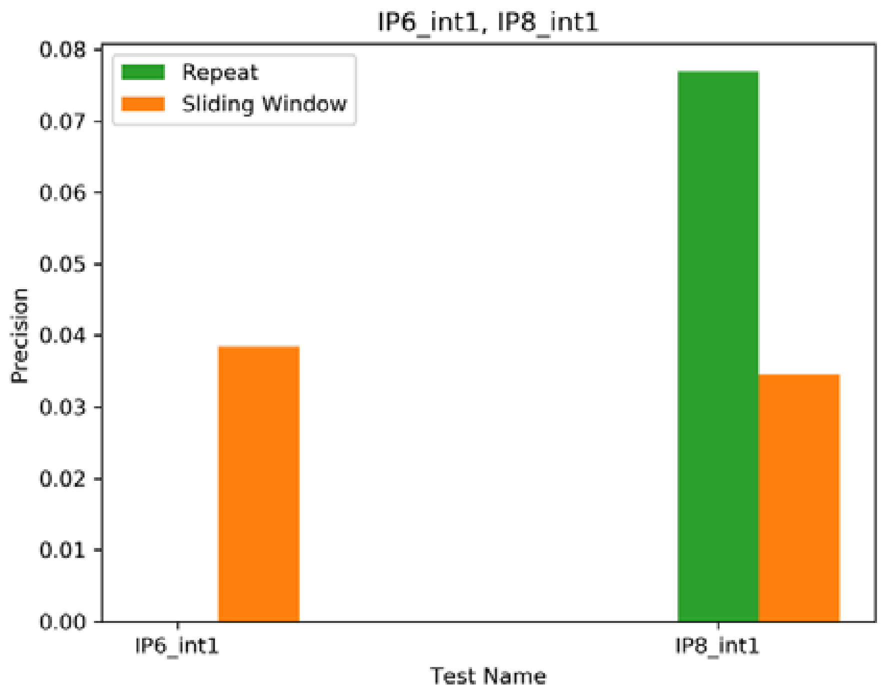


Figure 48. IP6_int1 and IP8_int1 attack detection results

The results of the IP6_int1 attack were as expected, with the repeat processing method failing to detect the attack, and the sliding window method successfully detecting the attack. The IP8_int1 attack was successfully detected by both detection algorithms. These results show that the expected performance failure of the repeat method is a possibility but is not guaranteed to occur.

6. Scalability Testing

Finally, a series of tests focusing on observing performance of the processing models as the observation capture size is scaled up were performed. The goal of these tests is to observe if the performance trends seen in the smaller-scale test generalize when data size is increased by one and two orders of magnitude. The best way to observe the outcome of these tests is to compare the number of false positive anomalies between the small-scale attack and the corresponding larger data scale tests. It is expected that the number of false positive anomalies are dependent on the number of comparisons between the training model and the observation models generated by the anomaly detection algorithm. Therefore, increasing the size of the observation data set will result in increases of the number of false positive anomalies for the repeat, sliding window, and alpha filter methods. Given that batch processing algorithm only conducts one comparison between the training and observation model, increasing the size of the data set should not affect the number of false positive anomalies. While the stream processing algorithm involves multiple comparisons between observation and training models, the observation model is never reset during the comparison, resulting in the statistical dilution of effect from the packets added later in the observation capture. Therefore, it is expected to have performance similar to the batch processing method. Figures 49–53 show the number of false positive anomalies generated for each of scalability tests. A discussion of the results shown in these figures is provided after Figure 53.

For all processing methods used in each test, increasing the size of the observation capture did not affect their ability to successfully detect the attack in any of the tests. If a given method detected an attack at the original scale, that attack was also detected in both the 10x and 100x scales. Likewise, if a method failed to detect an attack at the original scale, that attack remained undetected at both the 10x and 100x scales.

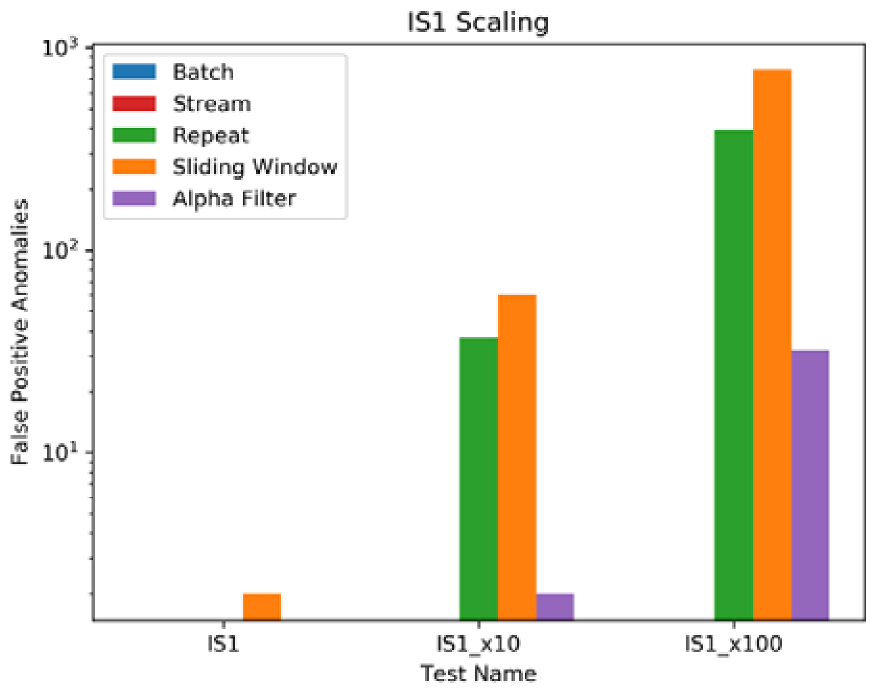


Figure 49. IS1 scaling false positive anomalies

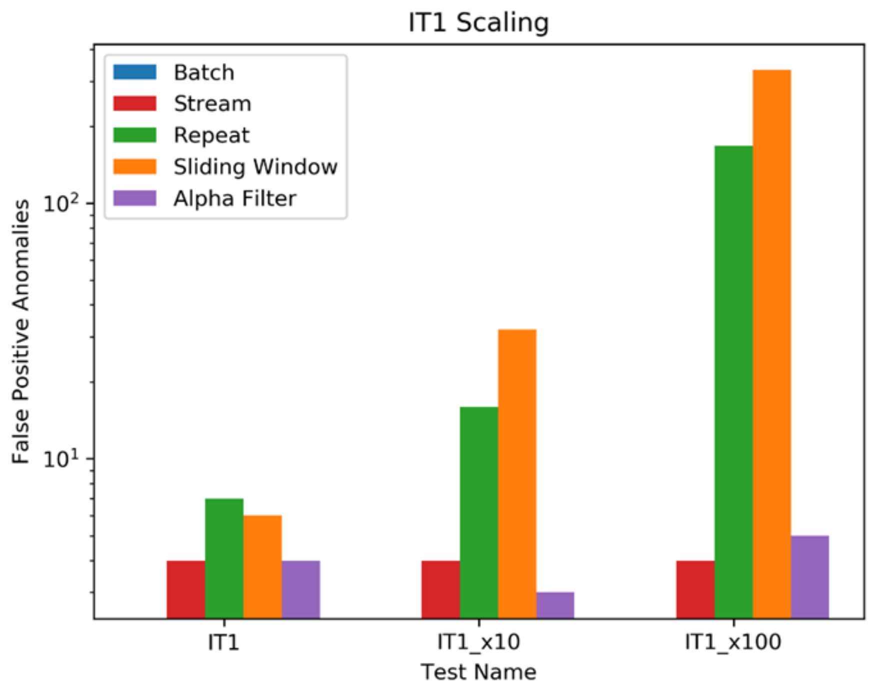


Figure 50. IT1 scaling false positive anomalies

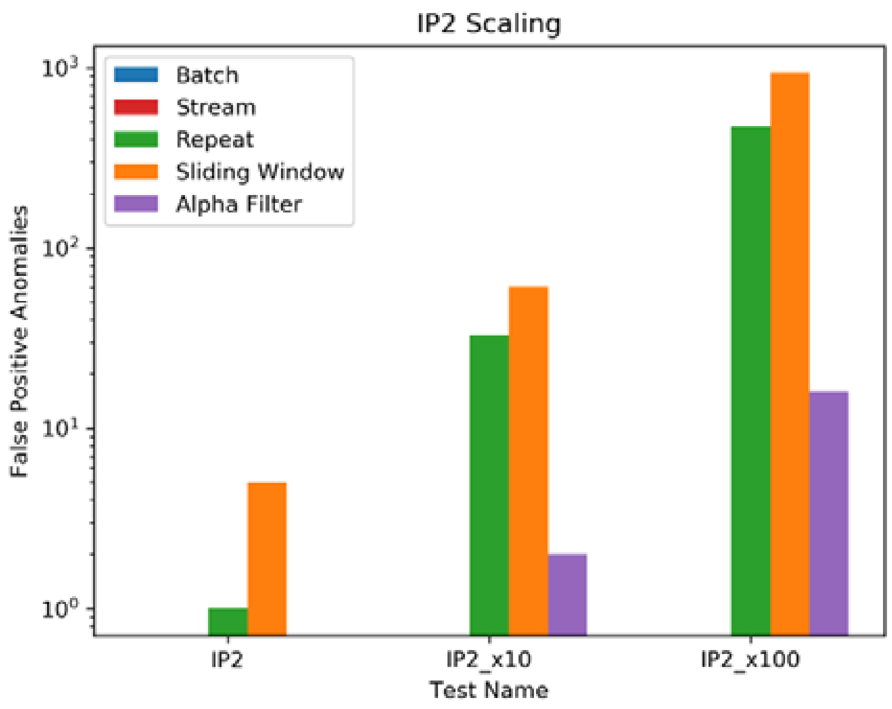


Figure 51. IP2 scaling false positive anomalies

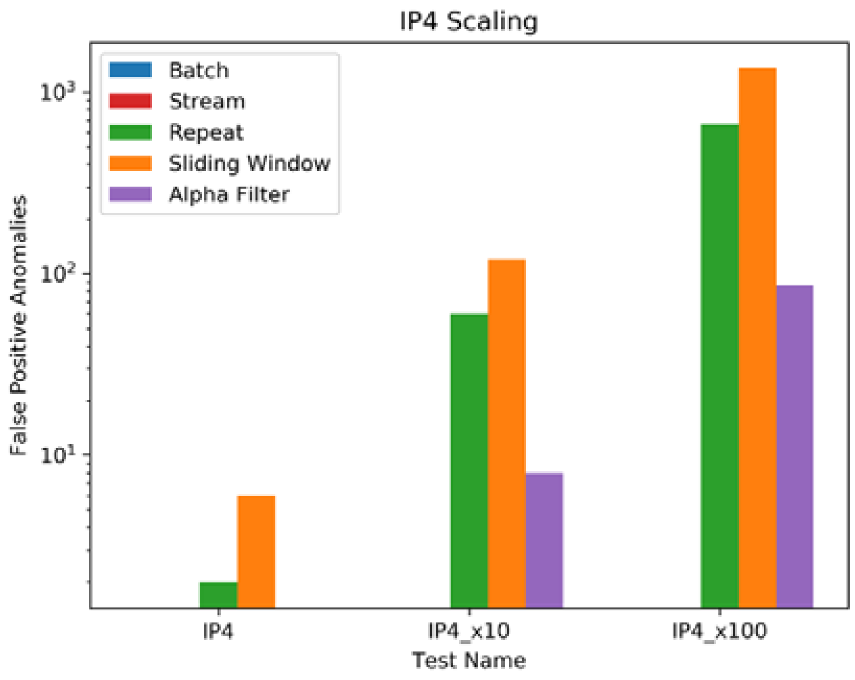


Figure 52. IP4 scaling false positive anomalies

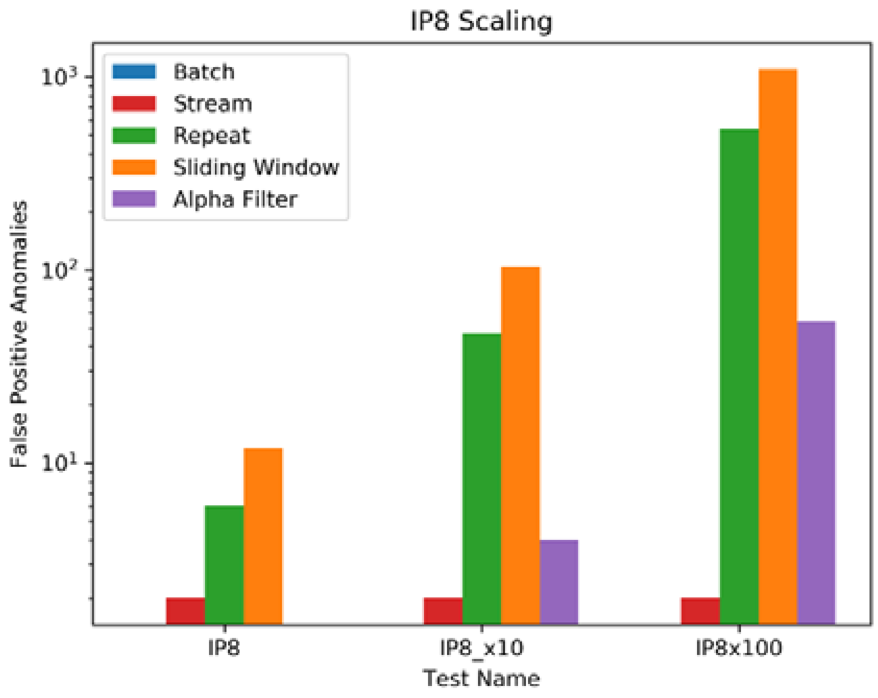


Figure 53. IP8 scaling false positive anomalies

Overall, the results of each of the scalability tests were as predicted. All attacks that were successfully detected in the smaller scale tests were also successfully detected in the scalability tests, and no attacks that failed to be detected in the small-scale tests were detected in the scalability tests. Additionally, the relationship between false positive anomalies and data size for each processing method is also shown to be as expected. These results show that the number of false positives from the batch and stream processing methods are mostly independent of the size of the data, while the number of false positives from the repeat, sliding window, and alpha filter are correlated to the size of the observation data set. As the size of the observation data set increased by an order of magnitude, so does the approximate number of false positive anomalies. There are two outliers to this trend. In Figure 50 the number of false positives generated by the alpha filter processing algorithm does not increase with the size of the observation data set, and in Figure 52 the number of false positives generated by the stream processing method does increase with the size of the observation data set. It is not clear as to why these two outliers exist, but every other scalability test shows the expected performance.

VI. CONCLUSIONS AND FUTURE WORK

The goal of this thesis was to develop a sequence-aware IDS for an ICS system that uses ENIP and CIP as its application layer protocols, and to test five anomaly detection algorithms for the sequence-aware IDS. A method for using discrete-time Markov chains (DTMC) to develop a sequence-aware IDS was described by Caselli et al. [4], but to the best of our knowledge, their method has not yet been implemented on an ICS using ENIP and CIP protocols. Additionally, they only described one method (batch processing) for processing traffic captures into observation DTMC models for attack detection, and this method struggles with detecting some attacks that only occur over a small portion of the capture. This work implemented four additional methods for processing an observation traffic capture into a DTMC observation model for anomaly detection, and compared the performance of all five of these anomaly detection methods under several attack scenarios. All five anomaly detection methods successfully detected attacks that resulted in invalid state or invalid transition anomalies. However, only the newly proposed repeat and sliding window methods were able to constantly detect attacks that resulted in localized probability anomalies, and only the sliding window method was shown to do so when the packets containing the attack were divided into separate observation models by the interval size parameter.

A. SUMMARY OF FINDINGS

A sequence-aware IDS that uses DTMC to model correct behavior of ICS components was implemented on the ICSICL testbed. This IDS was used to observe the performance of the batch processing method shown in [4] as well as the four processing methods proposed in this thesis (stream, repeat, sliding window, and alpha filter). These processing methods were used to detect attacks that would result in each of the three possible anomaly types, invalid state, invalid transition, and invalid probability. As expected, all processing methods were able to detect every attack that resulted in invalid state or invalid transition anomalies. Additionally, all methods were able to detect attacks that resulted in invalid probability anomalies if the attack was present throughout the entire capture. However, if the attack resulting in an invalid probability anomaly only occurred over a portion of the observation, the batch processing method would fail to detect the attack. Of the newly proposed methods, only the repeat and sliding window methods were able to constantly detect attacks. Furthermore, the sliding window method could do so when the packets containing the attack were divided into separate observation models by the interval size parameter.

1. Sequence-Aware IDS for ENIP and CIP Protocol

A sequence-aware IDS that uses a DTMC to model behavior was successfully implemented on an ICS using ENIP and CIP protocols. Chapters II and III described how the ENIP and CIP protocols encapsulate data for transmission between components of an ICS, and how those protocols are specifically used in the ICSICL testbed. Since the ICSICL implementation uses undocumented vendor-defined ENIP and CIP services, we had to engineer the attacks based on PCAP analysis of the manual manipulation of the ICSICL components. Although our implementation of the sequence-aware IDS for ICSICL was shown to successfully detect sequence-based attacks, it is not generalizable to all systems running ENIP and CIP.

2. Anomaly Detection Methods and Performance

Four of the five probability-based anomaly detection methods, i.e., batch, stream, repeat, and sliding window processing, performed close to expectations. Each of these methods succeeded in detecting the attacks they were expected to detect but failed to detect the attacks they were expected to fail to detect. The alpha filter processing method did not meet expectations as discussed in the future work section.

a. Batch Processing

The batch processing method successfully detected every invalid state and invalid transition attack. Additionally, no false positive anomalies occurred in any of these tests (IS1, IS2, IT1, IT2). However, the batch processing method showed inconsistent performance when detecting probability anomalies. While the IP2 and IP3 attacks were successfully detected, batch processing was the only detection method that failed to detected the IP1 attack. It also failed to detect any of the attacks that resulted in a localized anomaly (IP5-IP9).

However, the batch processing method was the only method to never generate a false positive anomaly. Therefore, while batch processing is the worst method for localized probability anomaly detection, it offers the best performance in all other scenarios. For the attacks it was able to detect, batch processing had the highest precision values, and did not observe a false positive anomaly for any of the attacks. If it were not for its inability to detect attacks that resulted in a localized anomaly, batch processing would be the best anomaly detection method.

b. Stream Processing

The stream processing method successfully detected every invalid state and invalid transition attack. Similar to the batch processing performance, no false positive anomalies occurred in the IS1 and IS2 tests. While stream processing for the IT1 and IT2 attacks did result in false positive anomalies, it had the second-best performance with four and three false positives respectively. The stream processing method also detected each attack in IP1-IP3 captures and did so with no false positive anomalies. However, out of the IP4-IP9 set of attacks, stream processing only detected the IP4 and IP5 attacks. This was expected due to the way stream processing builds the observation model.

Overall, stream processing shows a small performance increase over batch processing when detecting localized anomalies, but is only successful if the anomaly is early in the capture. With the exception of the IT1 and IT2 attacks, stream processing detected other attacks with no false positive anomalies.

c. Repeat Processing

The repeat processing method successfully detected every invalid state and invalid transition attack. No false positive anomalies occurred in the IS1 and IS2 attacks, but repeat processing had the highest rate of false positive anomalies for the IT1 and IT2 attacks, with seven and six false positive anomalies respectively.

The repeat processing method is the first anomaly detection method to successfully detect every IP1-IP9 attack. This shows that the process of creating multiple observation models out of a single capture results in consistent detection of localized anomalies. However, this comes at the cost of a large number of false positive anomalies. At best, repeat processing resulted in two false positive anomalies for the IP4 attack, but at its worst, resulted in thirteen false positive anomalies during the IP7 attack.

While repeat processing successfully detected the IP1-IP9 attacks, if the attack spanned the interval size used to divide the observation capture to generate separate observation models, it was possible to fail to detected the attack. This is shown with the IP6_int1 and IP8_int1 tests, where repeat processing failed to detect the IP6_int1 attack.

Overall, repeat processing showed successful performance for all types of attacks, with the exception of the IP6_int1 attack, but due to the high number of false positive anomalies and the inability to detect the IP6_int1 attack, repeat processing is not the ideal anomaly detection method.

d. Sliding Window Processing

As with all other anomaly detection methods, sliding window processing successfully detected the IS1, IS2, IT1, and IT2 attacks. However, it is the only method that reported false positive anomalies in the IS1 and IS2 attacks.

Similar to the repeat processing method, the sliding window anomaly detection was able to detected the attack in IP1-IP9 tests. However, it had the highest number of false positive anomalies of any anomaly detection method for every IP1-IP9 attack, with IP6 and IP7 resulting in seventeen false positive anomalies each. The sliding window processing method was the only method to successfully detect every attack that was tested, including the IP6_int1 and IP8_int1 attacks. However, this performance comes at the cost of a high number of false positive anomalies.

e. Alpha Filter Processing

The alpha filter processing method also successfully detected the IS1, IS2, IT2, and IT2 attacks. No false positive anomalies were observed in the IS1 or IS2 attacks, and four false positive anomalies were observed in the IT1 and IT2 attacks, which are comparable to the results of stream processing.

Alpha filter processing successfully detected each attack in IP1-IP3, and did so with no false positive anomalies. However, of the IP4-IP9 attacks, the alpha filter method only detected the IP5 attack. This shows similarity to the stream processing method, where localized anomaly can be detected if they occur early in the capture before being diluted by the continual additions of data to the observation model for stream processing.

The original expectations for the alpha filter processing methods were not met. It was expected that the smoothing effect of the alpha filter would reduce the number of false positives when compared to the repeat and sliding window methods, while still enabling detection of attacks that result in localized anomalies. However, the hyper-parameter tuning for Σ and α showed that in order to best detect localized anomalies, the value for α needs to approach zero. When $\alpha = 0$, the alpha filter method is functionally identical to the repeat processing method. Therefore, we concluded that the alpha filter method does not effectively detect attacks that result in a localized anomaly.

However, the alpha filter method did show a considerable reduction in the number of false positive anomalies when compared to the repeat and sliding window methods in the IP1-IP3 tests. This shows that applying a signal processing filter is a viable method to achieve some reduction in false positive anomalies.

B. FUTURE WORK

This section discusses potential enhancements to improve anomaly detection performance of the prototype IDS, as well as improvements to the ICSICL testbed that would allow more complex testing of anomaly detection in ENIP and CIP traffic.

1. Anomaly Detection Methods

Overall, the performance of the alpha filter detection method was disappointing. We anticipated that the alpha filter processing would have the same advantages of the repeat and sliding window processing methods (ability to detect localized anomalies), while mitigating their disadvantages (false positive anomalies). However, the smoothing effect of the alpha filter dominated any change in probability values caused by the localized anomalies, resulting in performance more comparable to the stream processing method. The alpha filter only tracks a smoothed mean of the distribution change. This research identifies the need to track velocity of change as well as the acceleration of the change in probability values. These more advanced filter architectures are known as alpha/beta or alpha/beta/gamma filters and should be investigated to potentially improve the alpha filter performance.

2. Testbed Improvements

Investigation of the ICSICL testbed revealed several factors that placed limitation on the prototype IDS implementation and experiment design. First, the current ladder logic program that manages the operations of the AFTL and DIOL does not send any component manipulation commands. Instead, the HMI sends a status request to the PLC every ~0.5 seconds, and the PLC responds with a message that contains the current status of AFTL and DIOL components. This has two effects on the IDS. First, automatic operation of the AFTL and DIOL is not supported, and therefore manual operation of AFTL and DIOL components is necessary to cause a change in the data portion of the network communications. Second, having communications updates every 0.5 seconds, whether a component has been manipulated or not, results in the time spent in each state having a large effect on the transition probabilities for that state. When combined with manual operation, this introduces significant variation in the value of transition probabilities across models, even those where the same procedure was performed. It is possible that the combination of these two factors resulted in the high numbers of false positive anomalies seen during testing.

The second limitation the current ICSICL implementation places on our IDS prototype is the use of vendor-defined service codes within the CIP protocol. These service codes are undocumented, which hampered our effort to create an IDS for a CIP implementation that is capable of observing system start-up and understanding how the communication manager of the CIP protocol choose to organize the location and sequence of the CIP service requests. This problem could be overcome by developing a ladder-logic program for ICSICL that only uses the standard CIP service codes, using an actual ICS for modelling and testing, or entering a service contract with Rockwell Automation to obtain the definitions of the vendor-defined classes.

Of these three options, developing a new ICSICL program offers additional benefits. A new ladder logic program for ICSICL could support automatic operation of the AFTL and DIOL components. This could produce two benefits. First, automatic operation of the AFTL and DIOL components would remove the human operator from the test generation process. Second, this automation would increase the number of tests that could be observed by the IDS, and reduce the variance in the time taken between each step of the NOP, resulting in having more data for analyzing the IDS performance, as well as eliminating any variance in transition probability values that were caused by the human operator.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] B. Galloway and G. P. Hancke, “Introduction to industrial control networks,” in *IEEE Communications Surveys & Tutorials*, vol. 15, no. 2, pp. 860–880, Second Quarter 2013, doi: 10.1109/SURV.2012.071812.00124

[2] K. Stouffer, V. Pillitteri, S. Lightman, M. Abrams, and A. Hahn, *Guide to Industrial Control Systems (ICS) Security*, National Institute of Standards and Technology, NIST SP 800-82r2, Jun. 2015. doi: 10.6028/NIST.SP.800-82r2

[3] R. M. Lee, M. J. Assante, T. Conway, *Analysis of the Cyber Attack on the Ukrainian Power Grid*. E-ISAC, Washington, DC, USA, Defense Use Case, March 18, 2016.

[4] M. Caselli, E. Zambon, and F. Kargl, “Sequence-aware intrusion detection in industrial control systems,” in *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security – CPSS ’15*. Singapore, Republic of Singapore, 2015, pp. 13–24, doi: 10.1145/2732198.2732200

[5] “HMS connecting devices, industrial network market shares 2019 according to HMS, HMS Networks, Halmstad, Sweden,” May 7, 2019. Accessed Sep. 3, 2020. [Online]. Available: https://www.hms-networks.com/news-and-insights/news-from-hms/2019/05/07/industrial-network-market-shares-2019-according-to-hms

[6] Y. Hu, A. Yang, H. Li, Y. Sun, and L. Sun, “A survey of intrusion detection on industrial control systems,” *International Journal of Distributed Sensor Networks*, vol. 14, no. 8, p. 155014771879461, Aug. 2018, doi: 10.1177/1550147718794615

[7] R. Langer “To Kill a Centrifuge, a Detailed Stuxnet Analysis.” Accessed Aug. 13, 2020. [Online]. Data-driven OT/ICS security, https://www.langner.com/to-kill-a-centrifuge/

[8] Caselli M., Zambon E., Petit J., Kargl F. (2015) “Modeling message sequences for intrusion detection in industrial control systems,” in Rice M., and Shenoi S. (eds.), *Critical Infrastructure Protection IX. ICCIP 2015. IFIP Advances in Information and Communication Technology,* vol 466. Springer, Cham. https://doi.org/10.1007/978-3-319-26567-4_4

[9] B. Ferling, J. Chromik, M. Caselli, and A. Remke, “Intrusion detection for sequence-based attacks with reduced traffic models,” in *Measurement, Modelling and Evaluation of Computing Systems*, vol. 10740, R. German, K.-S. Hielscher, and U. R. Krieger, Eds. Cham: Springer International Publishing, 2018, pp. 53–67.

[10] V. Schiffer, “Common industrial protocol (CIPTM) and the family of CIP networks,” in *Industrial Communication Technology Handbook*, 2nd ed., R. Zurawski, Ed. CRC Press, 2017, pp. 9–1-9–100.

[11] R. Zurawski, Ed., *Industrial Communication Technology Handbook*, 2nd ed. Boca Raton, London, New York: CRC Press, Taylor & Francis Group, 2015.

[12] “CIP networks library volume 2,” EtherNet/IP Adaptation of CIP vol. 2, ODVA Inc, 2017.

[13] “CIP networks library volume 1,” Common Industrial Protocol vol. 1, ODVA Inc., 2017.

[14] N. H. Desso, “Designing a machinery control system (MCS) security testbed,” M.S. thesis, Dept. of Computer Science, Naval Postgraduate School, Monterey, CA, 2014.

[15] O. C. Ibe, *Markov processes for stochastic modeling*, 2nd ed. Amsterdam, Netherlands: Elsevier, 2013.

[16] D. Hadžiosmanović, R. Sommer, E. Zambon, and P. H. Hartel, “Through the eye of the PLC: semantic security monitoring for industrial processes,” in *Proceedings of the 30th Annual Computer Security Applications Conference on ACSAC ‘14*, New Orleans, Louisiana, 2014, pp. 126–135, doi: 10.1145/2664243.2664277

[17] Center for Cybersecurity and Cyber Operations, *ICSICL-Initial PLC Configuration and Testing*, Monterey, CA: Naval Postgraduate School, 2017.

[18] I. A. Witten, E. Frank, M. A. Hall, and J. Pal, *Data Mining: Practical Machine Learning Tools and Techniques*, 4th ed. Cambridge, MA, USA: Morgan Kaufmann, 2017.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California